US012165038B2

(12) **United States Patent**
Lo et al.

(10) **Patent No.: US 12,165,038 B2**
(45) **Date of Patent: Dec. 10, 2024**

(54) ADJUSTING ACTIVATION COMPRESSION FOR NEURAL NETWORK TRAINING

(71) Applicant: **Microsoft Technology Licensing, LLC**, Redmond, WA (US)

(72) Inventors: **Daniel Lo**, Bothell, WA (US); **Bita Darvish Rouhani**, Bellevue, WA (US); **Eric S. Chung**, Woodinville, WA (US); **Yiren Zhao**, Cambridge (GB); **Amar Phanishayee**, Seattle, WA (US); **Ritchie Zhao**, Ithaca, NY (US)

(73) Assignee: **Microsoft Technology Licensing, LLC**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1271 days.

(21) Appl. No.: **16/276,395**

(22) Filed: **Feb. 14, 2019**

(65) **Prior Publication Data**

US 2020/0264876 A1 Aug. 20, 2020

(51) **Int. Cl.**
G06N 3/08 (2023.01)
G06F 9/30 (2018.01)
(Continued)

(52) **U.S. Cl.**
CPC ***G06N 3/063*** (2013.01); ***G06F 9/30025*** (2013.01); ***G06F 18/217*** (2023.01); ***G06N 3/084*** (2013.01)

(58) **Field of Classification Search**
CPC G06N 3/08; G06N 3/063; G06N 20/00; G06N 3/084; G06N 3/0895; G06N 3/09;
(Continued)

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,283,559	A	2/1994	Kalendra et al.
6,144,977	A	11/2000	Giangarra et al.

(Continued)

FOREIGN PATENT DOCUMENTS

CN	107636697	1/2018

OTHER PUBLICATIONS

Drumond et al., Training DNNs with Hybrid Block Floating Point, arXIV, Dec. 2, 2018, pp. 1-11 (Year: 2018).*

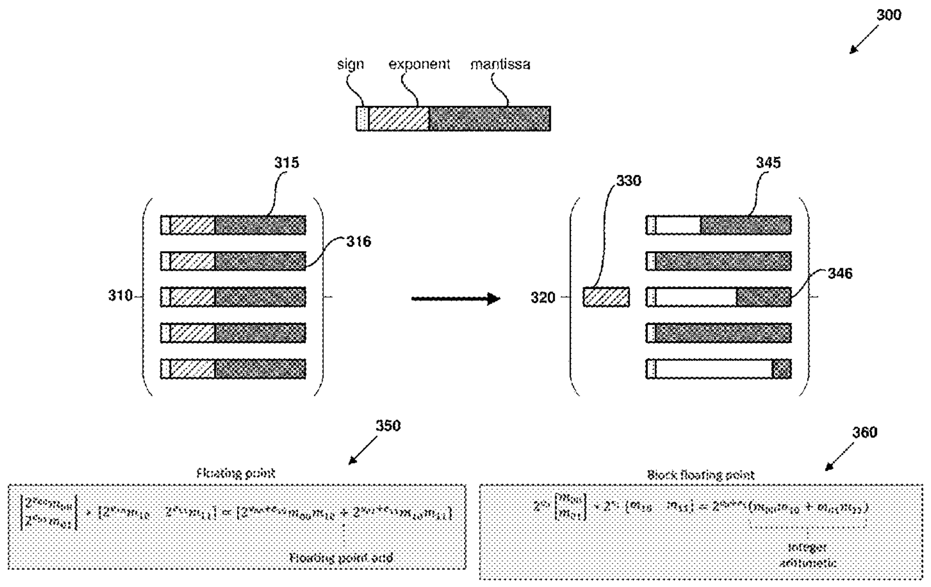
(Continued)

Primary Examiner — Paulinho E Smith
(74) *Attorney, Agent, or Firm* — Klarquist Sparkman, LLP

(57) **ABSTRACT**

Apparatus and methods for training a neural network accelerator using quantized precision data formats are disclosed, and, in particular, for adjusting floating-point formats used to store activation values during training. In certain examples of the disclosed technology, a computing system includes processors, memory, and a floating-point compressor in communication with the memory. The computing system is configured to produce a neural network comprising activation values expressed in a first floating-point format, select a second floating-point format for the neural network based on a performance metric, convert at least one of the activation values to the second floating-point format, and store the compressed activation values in the memory. Aspects of the second floating-point format that can be adjusted include the number of bits used to express mantissas, exponent format, use of non-uniform mantissas, and/or use of outlier values to express some of the mantissas.

21 Claims, 21 Drawing Sheets



(51) **Int. Cl.**

G06F 18/21	(2023.01)
G06N 3/063	(2023.01)
G06N 3/084	(2023.01)

(58) **Field of Classification Search**
CPC G06N 3/091; G06N 3/092; G06N 3/094; G06N 3/096; G06N 3/098; G06N 3/0985
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,708,068	B1	3/2004	Sakaue	
8,319,738	B2	11/2012	Taylor	
10,167,800	B1	1/2019	Chung et al.	
10,366,217	B2	7/2019	Finzi et al.	
11,537,859	B2 *	12/2022	Cassidy	G06N 3/063
11,604,647	B2 *	3/2023	Sun	G06F 17/18
2004/0041842	A1	3/2004	Lippincott	
2006/0209041	A1	9/2006	Studt et al.	
2007/0258641	A1	11/2007	Srinivasan et al.	
2008/0282046	A1	11/2008	Yuuki	
2011/0154006	A1	6/2011	Natu et al.	
2013/0222320	A1	8/2013	Huang et al.	
2014/0289445	A1	9/2014	Savich	
2015/0084900	A1	3/2015	Hodges et al.	
2016/0070414	A1	3/2016	Shukla et al.	
2016/0098149	A1	4/2016	Baumegatner	
2016/0328646	A1	11/2016	Lin et al.	
2018/0157465	A1	6/2018	Bittner et al.	
2018/0157899	A1 *	6/2018	Xu	G06N 3/08
2018/0322607	A1 *	11/2018	Mellempudi	G06F 7/5443
2018/0341857	A1 *	11/2018	Lee	G06N 3/04
2019/0075301	A1 *	3/2019	Chou	G06N 3/045
2019/0386717	A1	12/2019	Shattil	
2020/0042287	A1	2/2020	Chalamalasetti et al.	
2020/0202201	A1	6/2020	Shirahata et al.	
2020/0210840	A1 *	7/2020	Darvish Rouhani	G06N 3/082
2020/0272213	A1	8/2020	Shidharan et al.	

OTHER PUBLICATIONS

Chakrabarti, et al., "Backprop with Approximate Activations for Memory-efficient Network Training", In Journal of Computing Research Repository, Jan. 23, 2019, 09 Pages.
Krishnamoorthi, Raghuraman, "Quantizing Deep Convolutional Networks for Efficient Inference: A whitepaper", In Journal of Computing Research Repository, Jun. 21, 2018, 36 Pages.
Park, et al., "Value-Aware Quantization for Training and Inference of Neural Networks", In Proceedings of the European Conference on Computer Vision, Sep. 8, 2018, pp. 608-624.
"International Search Report and Written Opinion Issued in PCT Application No. PCT/US2020/015765", Mailed Date: May 15, 2020, 14 Pages.
Wang, et al., "Training Deep Neural Networks with 8-bit Floating Point Numbers", In Journal of Computing Research Repository, Dec. 19, 2018, 11 Pages.
"Final Office Action Issued in U.S. Appl. No. 16/237,308", Mailed Date: Oct. 6, 2022, 32 Pages.
Drumond, et al., "End-to-End DNN Training with Block Floating Point Arithmetic", In Repository of arXiv:1804.01526v2, Apr. 9, 2018, 9 Pages.
Drumond, et al., "Training DNNs with Hybrid Block Floating Point", In Repository of arXiv:1804.01526v4, Dec. 2, 2018, 11 Pages.
Han, et al., "Learning Both Weights and Connections for Efficient Neural Networks", In Repository of arXiv:1506.02626v3, Oct. 30, 2015, 9 Pages.
"International Search Report and Written Opinion Issued in PCT Application No. PCT/US2019/066676", Mailed Date: Apr. 15, 2020, 15 Pages.
Raghavan, et al., "Bit-Regularized Optimization of Neural Nets", In Repository of arXiv:1708.04788, Aug. 16, 2017, 11 Pages.
Song, et al., "Computation Error Analysis of Block Floating Point Arithmetic Oriented Convolution Neural Network Accelerator Design", In Repository of arXiv:1709.07776v2, Nov. 24, 2017, 8 Pages.
Wiedemann, et al., "Entropy-Constrained Training of Deep Neural Network", In Repository of arXiv:1812.07520v2, Dec. 19, 2019, 8 Pages.
Anonymous, Artificial Intelligence Index 2017 Annual Report, Nov. 2017, 101 pages.
Baydin et al., "Automatic Differentiation in Machine Learning: a Survey," Journal of Machine Learning Research 18 (2018), Feb. 5, 2018, 43 pages (also published as arXiv:1502.05767v4 [cs.SC] Feb. 5, 2018).
Bulò et al., "In-Place Activated BatchNorm for Memory-Optimized Training of DNNs," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Jun. 2018, pp. 5639-5647 (also published as arXiv:1712.02616 [cs.CV]).
Burger, "Accelerating Persistent Neural Networks at Datacenter Scale," Microsoft Corporation, 52 pp. accessed Apr. 18, 2018, available at: https://www.microsoft.com/en-us/research/blog/microsoft-unveils-project-brainwave/.
Burger, "Microsoft Unveils Project Brainwave for Real-Time AI," Microsoft Corporation, 3 pp (Aug. 18, 2018).
Chen et al., "Compressing Neural Networks with the Hashing Trick," In International Conference on Machine Learning, pp. 2285-2294, 2015 (also cited as arXiv:1504.04788v1 [cs.LG] Apr. 19, 2015).
Chiu et al., State-of-the-art Speech Recognition with Sequence-to-Sequence Models. CoRR, abs/1712.01769, 2017 (also cited as arXiv:1712.01769v6 [cs.CL] Feb. 23, 2018).
Chung et al., "Serving DNNs in Real Time at Datacenter Scale with Project Brainwave," IEEE Micro Pre-Print, 11 pages accessed Apr. 4, 2018, available at https://www.microsoft.com/en-us/research/uploads/prod/2018/03/mi0218_Chung-2018Mar25.pdf, also published as "Serving DNNs in Real Time at Datacenter Scale with Project Brainwave," IEEE Micro, vol. 38, Issue 2, Mar./Apr. 2018.
Colah, "Understanding LSTM Networks," posted on Aug. 27, 2015, 13 pages.
Courbariaux et al., "Low precision arithmetic for deep learning," also available as arXiv:1412.7024v1, Dec. 2014.
Courbariaux et al., "Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to +1 or −1," arXiv preprint arXiv:1602.02830v3, Mar. 2016, 11 pages.
Courbariaux et al., "Binaryconnect: Training Deep Neural Networks with Binary Weights During Propagations," In Proceedings of the 28th International Conference on Neural Information Processing Systems, vol. 2, Dec. 2015, 9 pages.
Courbariaux et al., "Training Deep Neural Networks with Low Precision Multiplications," Sep. 23, 2015, 10 pages.
CS231n Convolutional Neural Networks for Visual Recognition, downloaded from cs231n.github.io/optimization-2, Dec. 20, 2018, 9 pages.
Denil et al., Predicting Parameters in Deep Learning, In Advances in Neural Information Processing Systems, Dec. 2013, pp. 2148-2156.
Elam et al., "A Block Floating Point Implementation for an N-Point FFT on the TMS320C55x DSP," Texas Instruments Application Report SPRA948, Sep. 2003, 13 pages.
"FFT/IFFT Block Floating Point Scaling," Altera Corporation Application Note 404, Oct. 2005, ver. 1.0, 7 pages.
Goodfellow et al., "Deep Learning," downloaded from http://www.deeplearningbook.org/ on May 2, 2018, (document dated 2016), 766 pages.
Gomez, "Backpropogating an LSTM: A Numerical Example," Apr. 18, 2016, downloaded from medium.com/@aidangomez/let-s-do-this-f9b699de31d9, Dec. 20, 2018, 8 pages.
Gupta et al., "Deep Learning with Limited Numerical Precision," Feb. 9, 2015, 10 pages.
Han et al., "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," arXiv preprint arXiv:1510.00149v5 [cs:CV], Feb. 15, 2016, 14 pages.

(56) **References Cited**

OTHER PUBLICATIONS

Hassan et al., "Achieving Human Parity on Automatic Chinese to English News Translation," CoRR, abs/1803.05567, 2018 (also published as arXiv:1803.05567v2 [cs.CL] Jun. 29, 2018).
He et al., "Deep Residual Learning for Image Recognition," arXiv preprint arXiv:1512.03385v1 [cs.CV] Dec. 10, 2015.
Ioffe et al., "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," arXiv:1502.03167v3 [cs.LG], Mar. 2015, 11 pages.
Jain et al., "Gist: Efficient Data Encoding for Deep Neural Network Training," 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture, Jun. 2018, 14 pages.
Karl N's Blog., "Batch Normalization—What the hey?," Posted on Jun. 7, 2016, downloaded from gab41.lab41.org/batch-normalization-what-the-hey-d480039a9e3b, Jan. 9, 2019, 7 pages.
Kevin's Blog, "Deriving the Gradient for the Backward Pass of Batch Normalization," Posted on Sep. 14, 2016, downloaded from kevinzakka.github.io/2016/09/14/batch_normalization/, Jan. 9, 2019, 7 pages.
Köster et al., "Flexpoint: An Adaptive Numerical Format for Efficient Training of Deep Neural Networks," In Advances in Neural Information Processing Systems, pp. 1742-1752, 2017 (also published as arXiv:1711.02213v2 [cs:LG] Dec. 2, 2017).
Kratzert's Blog, "Understanding the backward pass through Batch Normalization Layer," Posted on Feb. 12, 2016, downloaded from kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-bathch- nor . . . on Jan. 9, 2019, 17 pages.
Langhammer et al., "Floating-Point DSP Block Architecture for FPGAs," Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Feb. 2015, pp. 117-125.
Le et al., "Neural Architecture Search with Reinforcement Learning," PowerPoint presentation, 37 pages.
Le, "A Tutorial on Deep Learning, Part 1: Nonlinear Classifiers and the Backpropagation Algorithm," Dec. 2015, 18 pages.
Le, "A Tutorial on Deep Learning, Part 2: Autoencoders, Convolutional Neural Networks and Recurrent Neural Networks," Oct. 2015, 20 pages.
Lecun et al., "Optimal Brain Damage," In Advances in Neural Information Processing Systems, Nov. 1989, pp. 598-605.
Li et al., "Stochastic Modified Equations and Adaptive Stochastic Gradient Algorithms," Proceedings of the 34th International Conference on Machine Learning, PMLR 70, 2017, 10 pages.
Li et al., "Ternary eight networks," arXiv preprint arXiv:1605.04711v2 [cs:CV] Nov. 19, 2016.
Lin et al., "Fixed Point Quantization of Deep Convolutional Networks," In International Conference on Machine Learning, pp. 2849-2858, 2016 (also published as arXiv:1511.06393v3 [cs:LG] Jun. 2, 2016).
Liu, "DARTS: Differentiable Architecture Search," arXiv:1806.09055v1 [cs.LG], Jun. 24, 2018, 12 pages.
Mellempudi et al., "Ternary Neural Networks with Fine-Grained Quantization," May 2017, 11 pages.
Mendis et al., "Helium: Lifting High-Performance Stencil Kernals from Stripped x86 Binaries to Halide DSL Code," Proceedings of the 36th ACM SIGPLAN Conference on Programming Languate Design and Implementation, Jun. 2015, pp. 391-402.
Mishra et al., "Apprentice: Using Knowledge Distillation Techniques to Improve Low-Precision Network Accuracy," arXiv preprint arXiv:1711.05852v1 [cs:LG] Nov. 15, 2017.
Muller et al., "Handbook of Floating-Point Arithmetic," Birkhäuser Boston (New York 2010), 78 pages including pp. 269-320.
Nielsen, "Neural Networks and Deep Learning," downloaded from http://neuralnetworksanddeeplearning.com/index.html on May 2, 2018, document dated Dec. 2017, 314 pages.
Nvidia. Nvidia tensorrt optimizer, https://developer.nvidia.com/tensorrt downloaded on Mar. 4, 2019, 9 pages.
Page, "Neural Networks and Deep Learning," www.cs.wise.edu/~dpage/cs760/, 73 pp.
Park et al., "Energy-efficient Neural Network Accelerator Based on Outlier-aware Low-precision Computation," 2018 ACM/IEEE 4th Annual International Symposium on Computer Architecture, Jun. 2018, pp. 688-698.
Rajagopal et al., "Synthesizing a Protocol Converter from Executable Protocol Traces," IEEE Transactions on Computers, vol. 40, No. 4, Apr. 1991, pp. 487-499.
Rajpurkar et al., "SQuAD: 100,000+ Questions for Machine Comprehension of Text," Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Nov. 2016, pp. 2383-2392.
Rastegari et al., "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," In Proceedings of 14th Annual European Conference on Computer Vision, pp. 525-542. Oct. 2016.
"Russakovsky et al., ""ImageNet Large Scale Visual Recognition Challenge,"" International Journal of Computer Vision (IJCV), vol. 115, Issue 3, Dec. 2015, pp. 211-252 (also published as zrXiv:1409.0575v3 [cs.CV] Jan. 30, 2015)."
Russinovich, "Inside the Microsoft FPGA-based Configurable Cloud," Microsoft Corporation, https://channel9.msdn.com/Events/Build/2017/B8063, 8 pp. (May 8, 2017).
Russinovich, "Inside the Microsoft FPGA-based Configurable Cloud," Microsoft Corporation, Powerpoint Presentation; 41 pp. (May 8, 2017).
Smith et al., "A Bayesian Perspective on Generalization and Stochastic Gradient Descent," 6th International Conference on Learning Representations, Apr.-May 2018, 13 pages.
Szegedy et al., "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," arXiv:1602.07261v2, Aug. 23, 2016, 12 pages.
Szegedy et al., "Rethinking the Inception Architecture for Computer Vision," arXiv:1512.00567v3 [cs.CV] Dec. 11, 2015, 10 pages.
Tensorflow-slim image classification model library. https://github.com/tensorflow/models/tree/master/research/slim, downloaded on Mar. 4, 2019, 8 pages.
TITU1994 Blog, "Neural Architecture Search with Controller RNN," downloaded from github.com/titu1994/neural-architecture-search on Jan. 9, 2019, 3 pages.
Vanhoucke et al., "Improving the speed of neural networks on CPUs," In Deep Learning and Unsupervised Feature Learning Workshop, Dec. 2011, 8 pages.
Vucha et al., "Design and FPGA Implementation of Systolic Array Architecture for Matrix Multiplication," International Journal of Computer Applications, vol. 26, No. 3, Jul. 2011, 5 pages.
Weinberger et al., "Feature Hashing for Large Scale Multitask Learning," In Proceedings of the 26th Annual International Conference on Machine Learning, Jun. 2009, 8 pages.
Wen et al., "Learning Structured Sparsity in Deep Neural Networks," In Advances in Neural Information Processing Systems, Dec. 2016, pp. 2074-2082 (also published as arXiv:1608.036654v4 [cs.NE] Oct. 18, 2016).
Wilkinson, "Rounding Errors in Algebraic Processes," Notes on Applied Science No. 32, Department of Scientific and Industrial Research, National Physical Laboratory (United Kingdom) (London 1963), 50 pages including pp. 26-33, 79-91, and 122-139.
Wired, "Microsoft's Internet Business Gets a New Kind of Processor," 11 pp. Apr. 19, 2018, available at: https://www.wired.com/2016/09/microsoft-bets-future-chip-reprogram-fly/.
Xiong et al., "Achieving Human Parity in Conversational Speech Recognition," arXiv:1610.05256v2 [cs:CL] Feb. 17, 2017, 13 pages.
Yeh, "Deriving Batch-Norm Backprop Equations," downloaded from chrisyeh96.github.io/2017/08/28/deriving-batchnorm-backprop on Dec. 20, 2018, 5 pages.
Zhou et al., "DoReFa-net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," arXiv:1606.06160v3 [cs.NE] Feb. 2, 2018, 13 pages.
Zoph et al., "Learning Transferable Architectures for Scalable Image Recognition," arXiv.1707.07012v1, Jul. 2017, 14 pages.
Zoph et al., "Neural Architecture Search with Reinforcement Learning," 5th International Conference on Learning Representations, Apr. 2017, 16 pages.
"Non-Final Office Action Issued in U.S. Appl. No. 16/237,308", Mailed Date: Feb. 3, 2022, 25 Pages.

(56) **References Cited**

OTHER PUBLICATIONS

Park et al., "Cell division: weight bit-width reduction technique for convolutional neural network hardware accelerators," Proceedings of the 24th Asia and South Pacific Design Automation Conference, document dated Jan. 21, 2019, pp. 286-291.
Park et al., "Energy-efficient Neural Network Accelerator Based on Outlier-aware Low-precision Computation," 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture, Jun. 1, 2018, pp. 688-709.
Zhao et al., "Improving Neural Network Quantization without Retraining using Outlier Channel Splitting," arXiv:1901.09504v2 [cs.LG], document dated Jan. 30, 2019, 10 pages.
"Final Office Action Issued in U.S. Appl. No. 16/237,308", Mailed Date: Aug. 9, 2023, 33 Pages.
"Non Final Office Action Issued in U.S. Appl. No. 16/237,308", Mailed Date: Feb. 3, 2023, 31 Page.
"Office Action Issued in Indian Patent Application No. 202147035445", Mailed Date: Feb. 16, 2023, 6 Pages.
Communication pursuant to Article 94(3) EPC, Received for European Application No. 20708371.8, mailed on Mar. 20, 2024, 08 pages.
Office Action Received for Chinese Application No. 202080014556.X, mailed on Apr. 15, 2024, 24 pages (English Translation Provided).
Office Action Received for European Application No. 19839023.9, mailed on Jan. 2, 2024, 09 pages.
Controller Calibration on Resistive Touch Screens, DMC is a touch screen manufacturer, DMC Co. LTD., 1,22,2019, 2 pages.
International Preliminary Report on Patentability, received in PCT/US2020/015765, Aug. 10, 2021, 10 pages.
International Preliminary Report on Patentability received in PCT/US2019/066676, Jun. 16, 2021, 12 pages.
International Search Report received in PCT/US2021/017406, Apr. 28, 2020, 10 pages.
International Preliminary Report on Patentability received in PCT/US2020/017406, Aug. 25, 2021, 9 pages.
Non-Final Office Action received in U.S. Appl. No. 16/286,098, filed Nov. 2, 2020, 10 pages.
Notice of Allowance received in U.S. Appl. No. 16/286,098, filed Feb. 24, 2021, 5 pages.
First Examination Report, received in India Application No. 202147036372, Feb. 13, 2023, 6 pages.
Drummond et al., "Training DNNs with Hybrid Block Floating Point," 32nd Conference on Neural Information Processing Systems, Montreal, Canada, 2018.
Applicant Initiated Interview Summary, U.S. Appl. No. 16/237,308, filed Jan. 4, 2023, 2 pages.
Applicant Initiated Interview Summary, U.S. Appl. No. 16/286,098, filed Jan. 29, 2021, 2 pages.

* cited by examiner

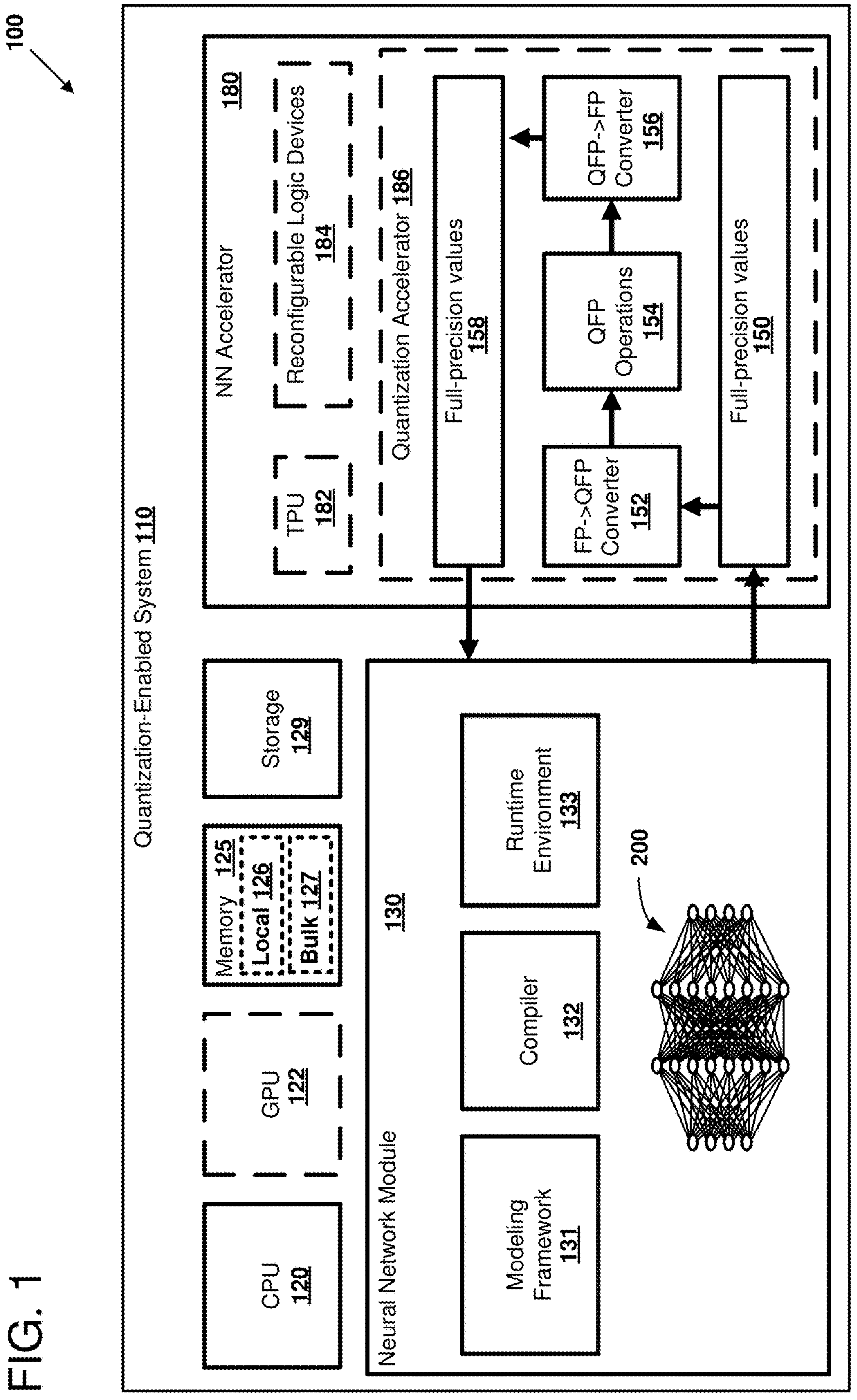

100
Quantization-Enabled System 110
CPU 120
GPU 122
Memory 125
Local 126
Bulk 127
Storage 129
NN Accelerator 180
TPU 182
Reconfigurable Logic Devices 184
Quantization Accelerator 186
Full-precision values 158
FP->QFP Converter 152
QFP Operations 154
QFP->FP Converter 156
Full-precision values 150
Neural Network Module 130
Modeling Framework 131
Compiler 132
Runtime Environment 133
200

FIG. 1

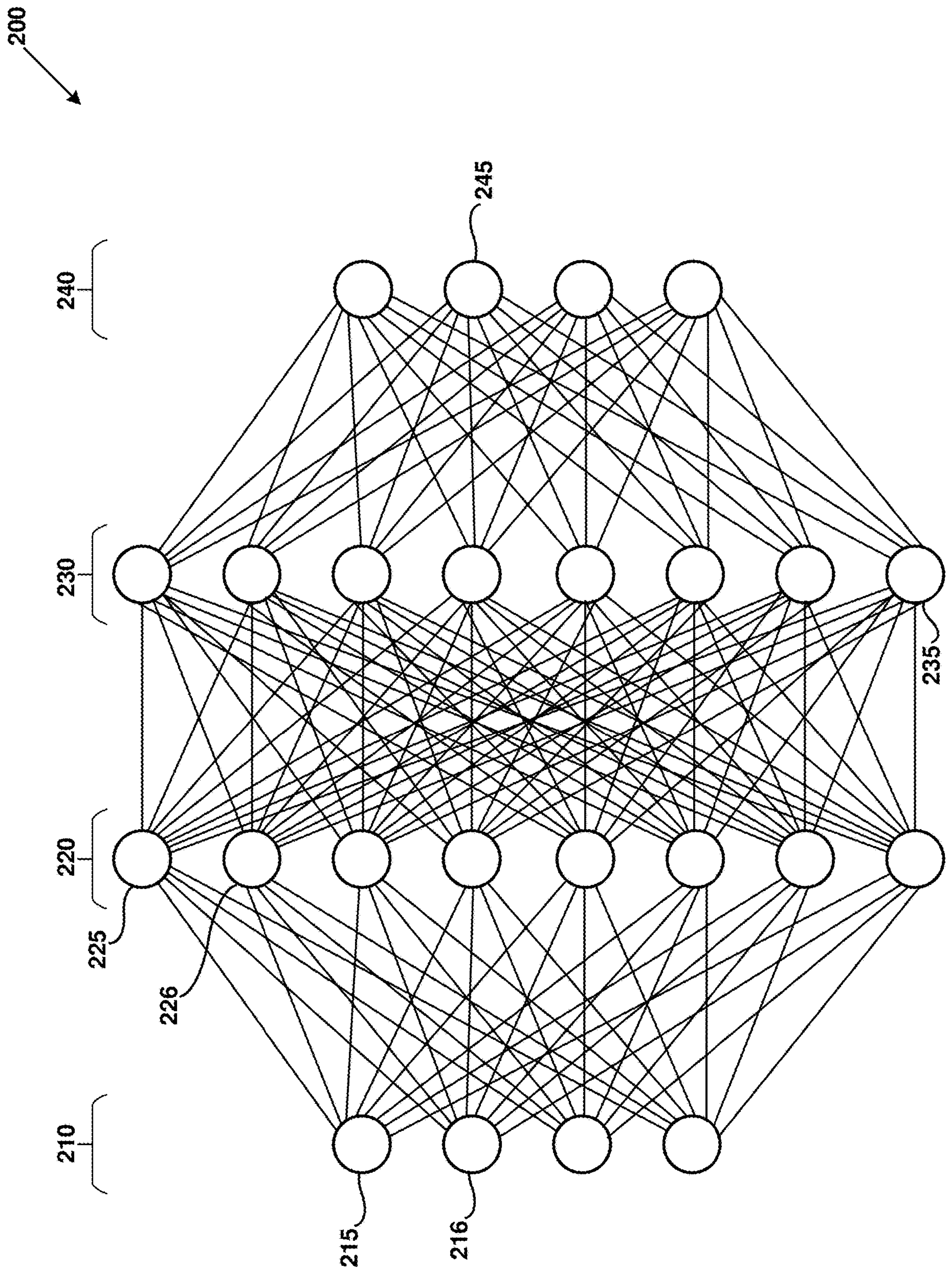


FIG. 2

FIG. 3

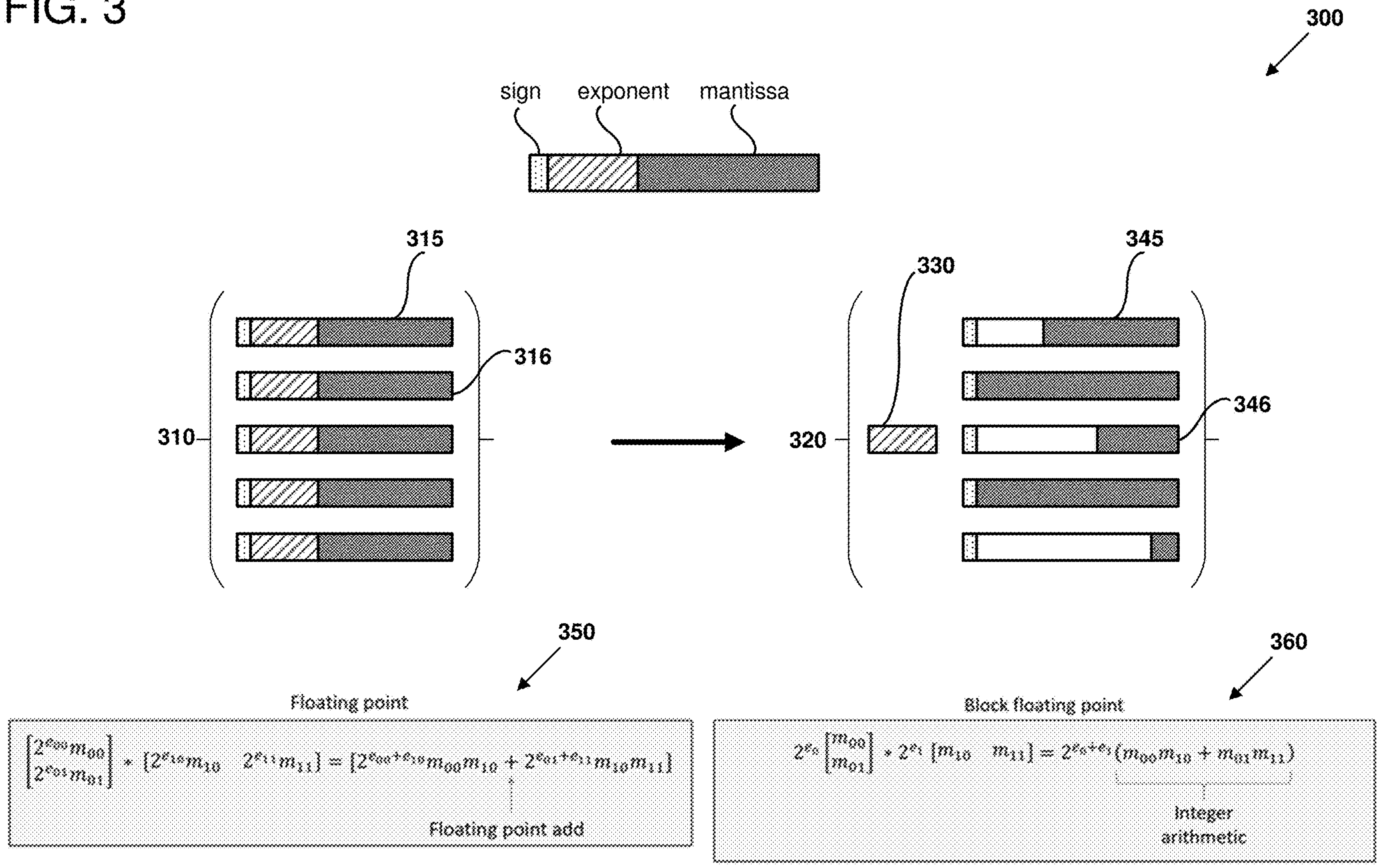

300
sign
exponent
mantissa
315
316
310
330
345
346
320
350
Floating point
$\begin{bmatrix} 2^{e_{00}} m_{00} \\ 2^{e_{01}} m_{01} \end{bmatrix} * \begin{bmatrix} 2^{e_{10}} m_{10} & 2^{e_{11}} m_{11} \end{bmatrix} = \begin{bmatrix} 2^{e_{00}+e_{10}} m_{00} m_{10} + 2^{e_{01}+e_{11}} m_{10} m_{11} \end{bmatrix}$
Floating point add
360
Block floating point
$2^{e_0} \begin{bmatrix} m_{00} \\ m_{01} \end{bmatrix} * 2^{e_1} \begin{bmatrix} m_{10} & m_{11} \end{bmatrix} = 2^{e_0+e_1} \underbrace{(m_{00} m_{10} + m_{01} m_{11})}$
Integer arithmetic

FIG. 4

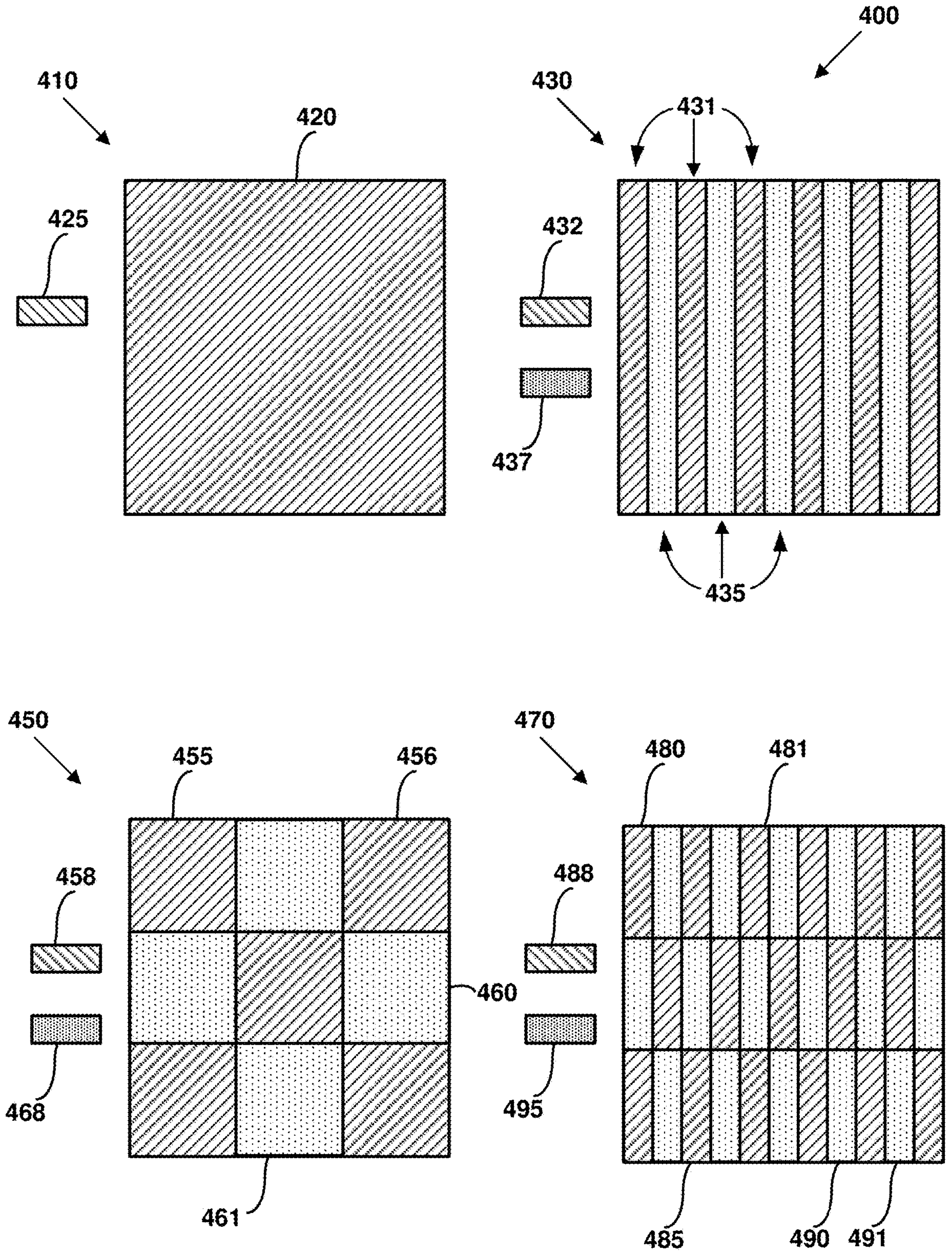

400
410
420
425
430
431
432
437
435
450
455
456
458
460
468
461
470
480
481
488
495
485
490
491

FIG. 5

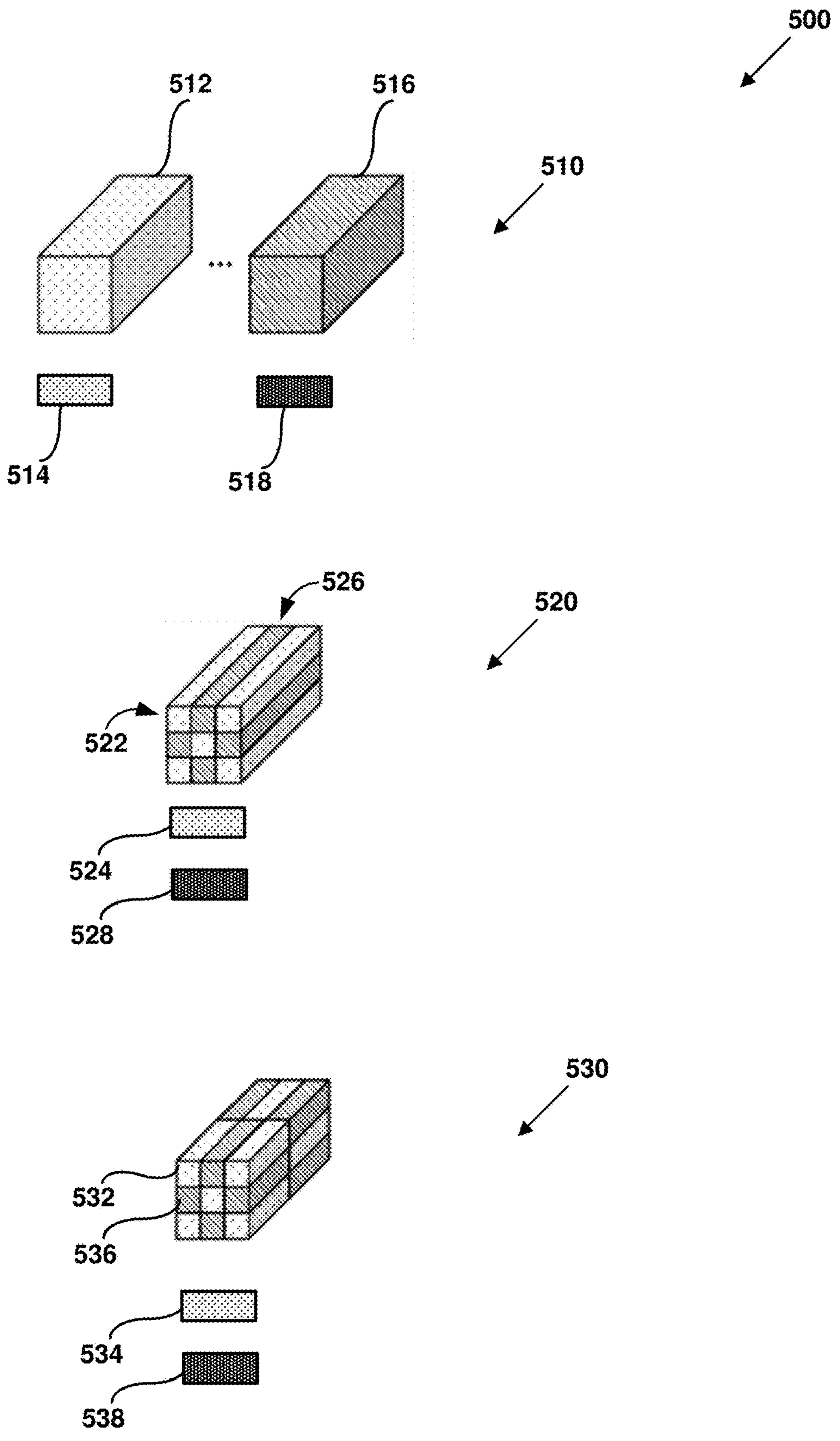
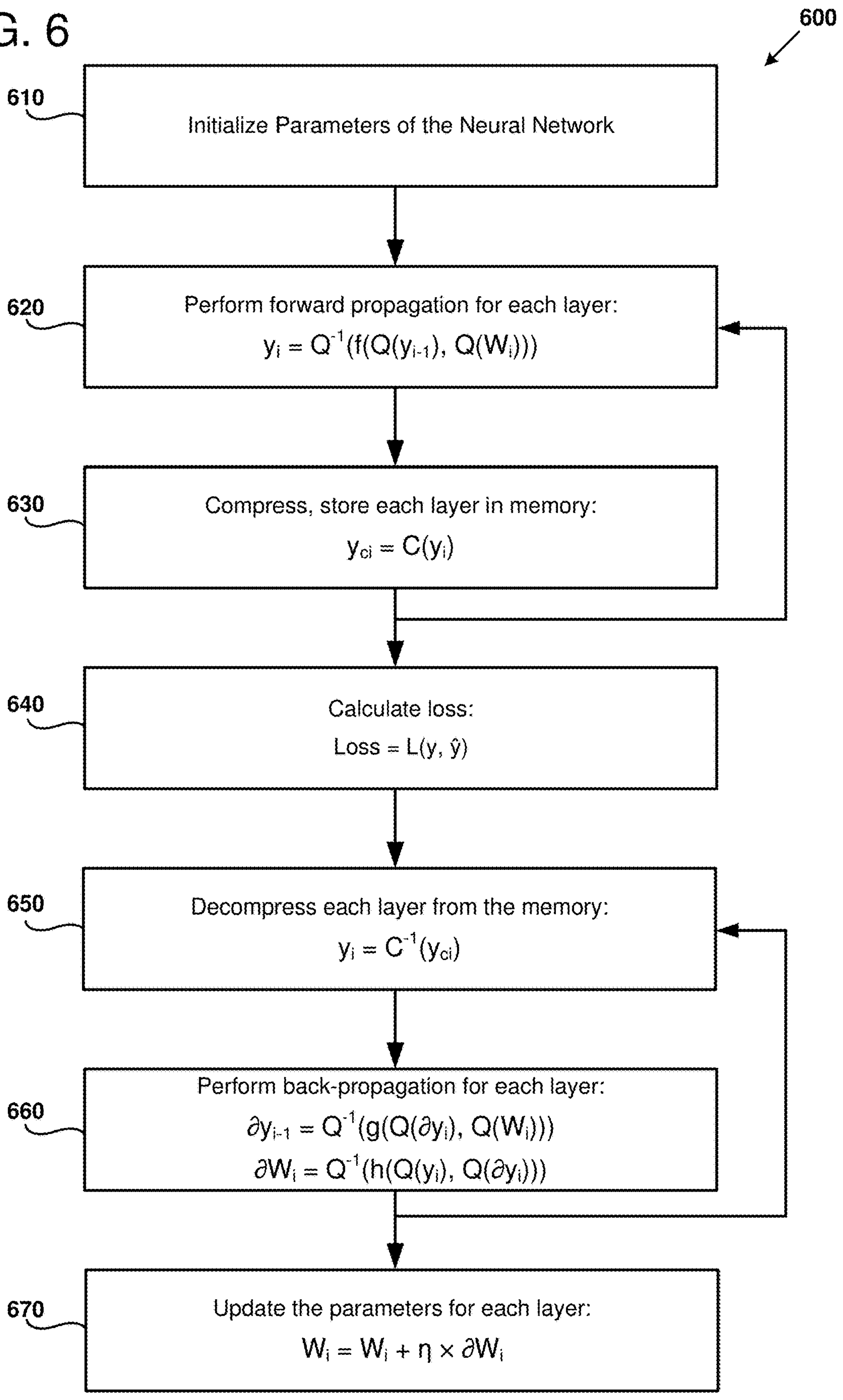


FIG. 6


600
610
Initialize Parameters of the Neural Network
620
Perform forward propagation for each layer:
$y_i = Q^{-1}(f(Q(y_{i-1}), Q(W_i)))$
630
Compress, store each layer in memory:
$y_{ci} = C(y_i)$
640
Calculate loss:
$\text{Loss} = L(y, \hat{y})$
650
Decompress each layer from the memory:
$y_i = C^{-1}(y_{ci})$
660
Perform back-propagation for each layer:
$\partial y_{i-1} = Q^{-1}(g(Q(\partial y_i), Q(W_i)))$
$\partial W_i = Q^{-1}(h(Q(y_i), Q(\partial y_i)))$
670
Update the parameters for each layer:
$W_i = W_i + \eta \times \partial W_i$

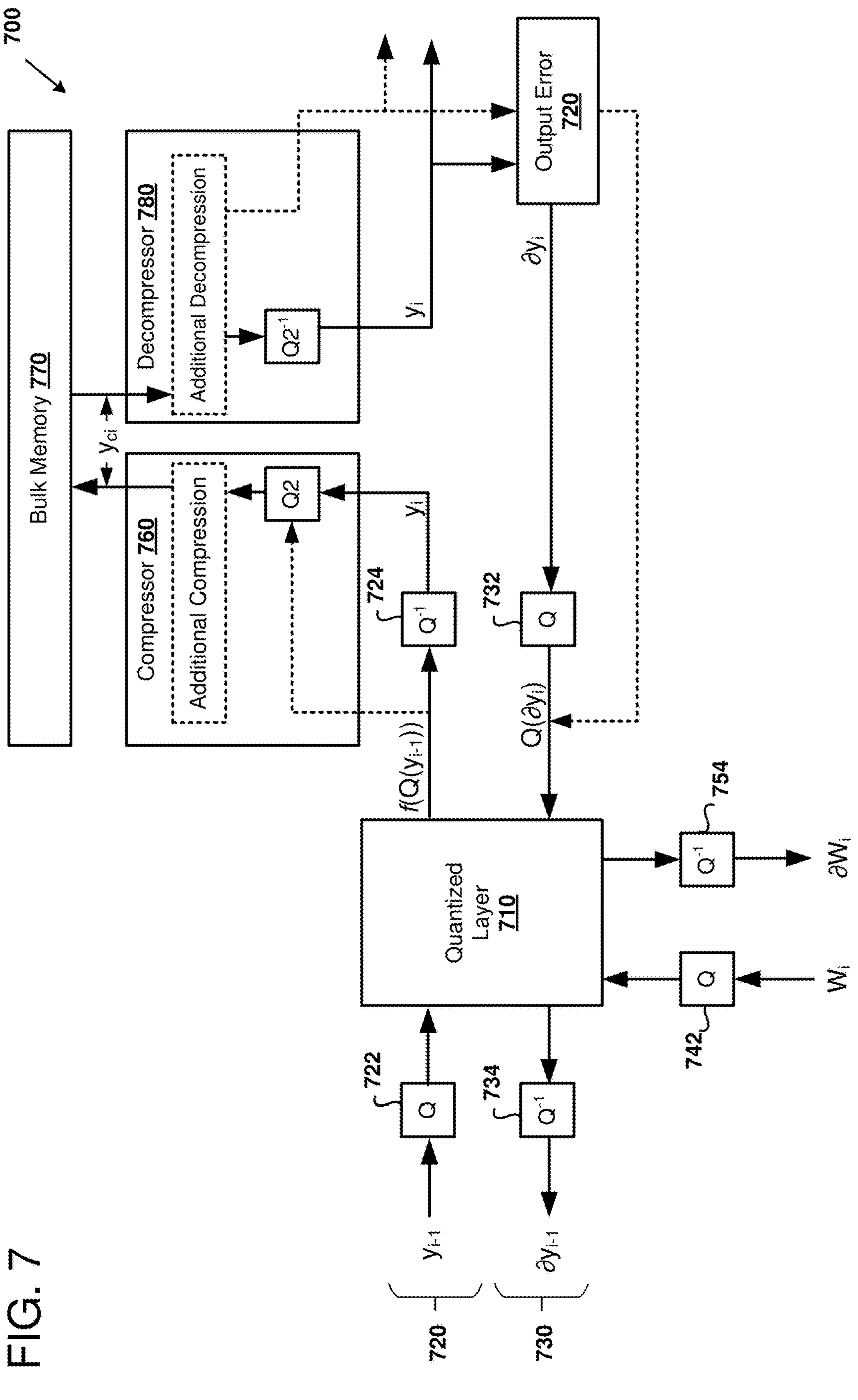

700
Bulk Memory 770
Compressor 760
Additional Compression
Q2
Decompressor 780
Additional Decompression
$Q2^{-1}$
y_{ci}
y_i
y_i
Output Error 720
∂y_i
724
Q^{-1}
732
Q
$f(Q(y_{i-1}))$
$Q(\partial y_i)$
Quantized Layer 710
754
Q^{-1}
∂W_i
Q
W_i
742
722
Q
734
Q^{-1}
y_{i-1}
∂y_{i-1}
720
730

FIG. 7

FIG. 8

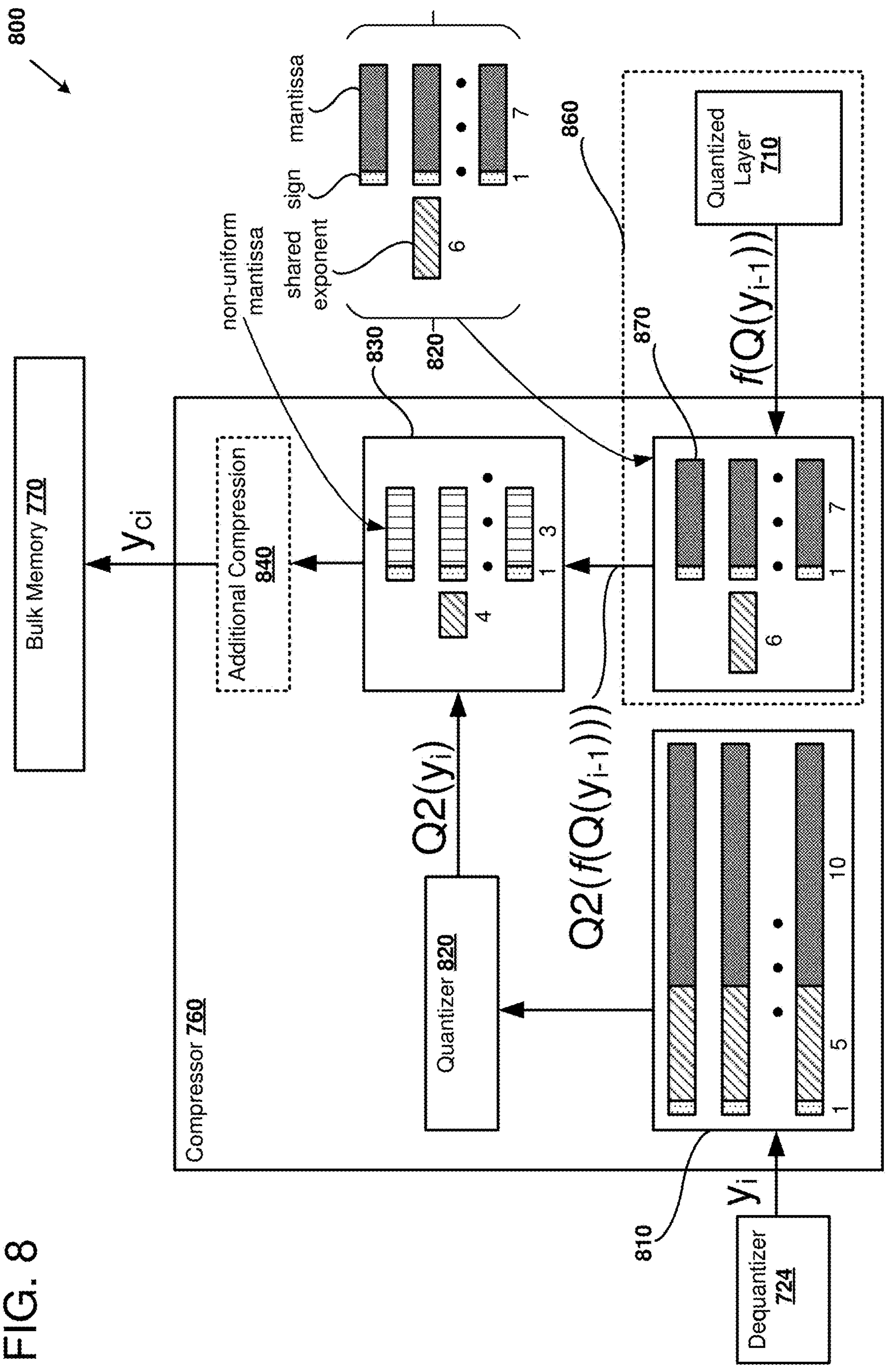

800
Bulk Memory 770
y_{ci}
Compressor 760
Additional Compression 840
non-uniform mantissa
830
820
shared exponent
sign
mantissa
6
1
7
4
3
860
870
Quantized Layer 710
$f(Q(y_{i-1}))$
$Q2(y_i)$
$Q2(f(Q(y_{i-1})))$
Quantizer 820
5
10
810
y_i
Dequantizer 724

FIG. 9

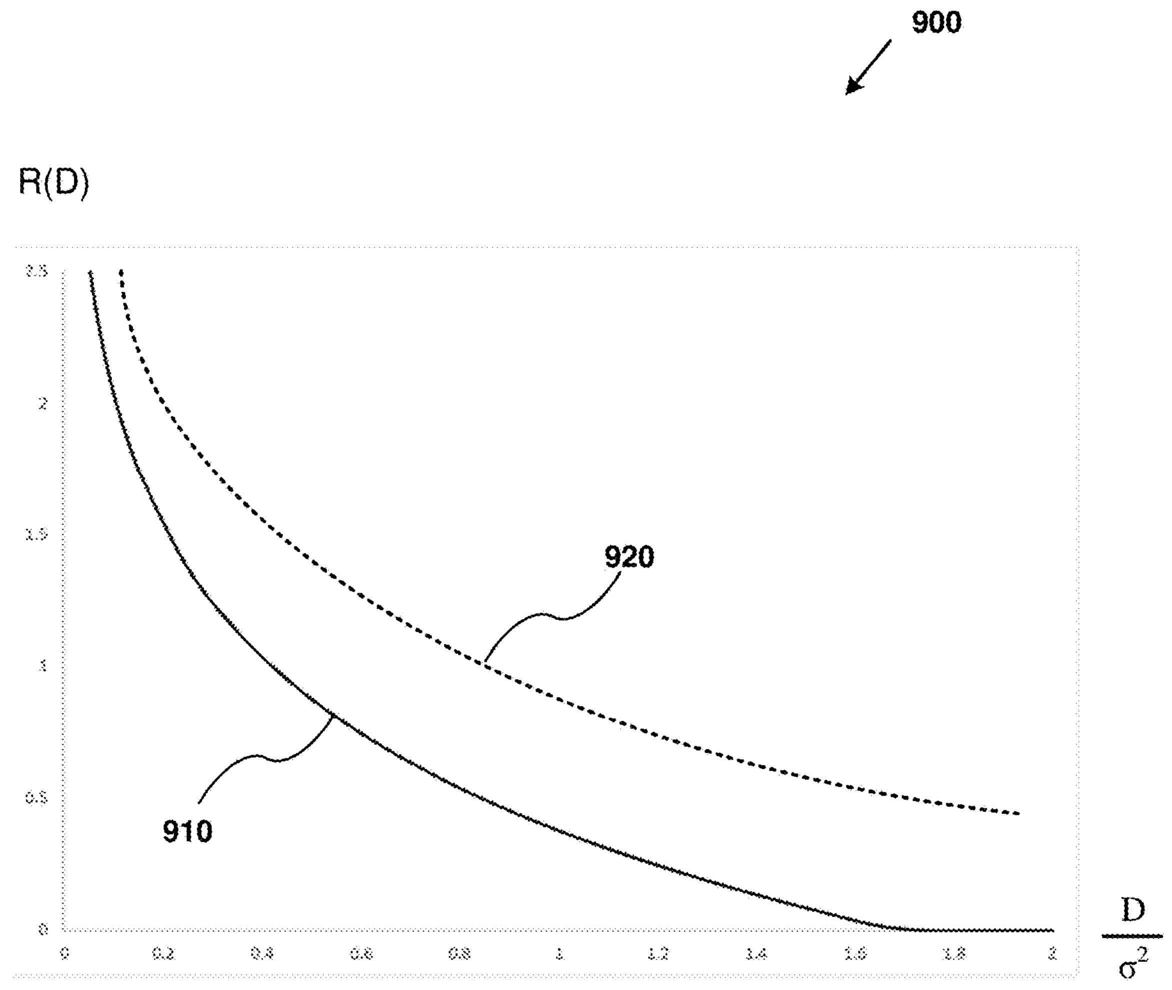
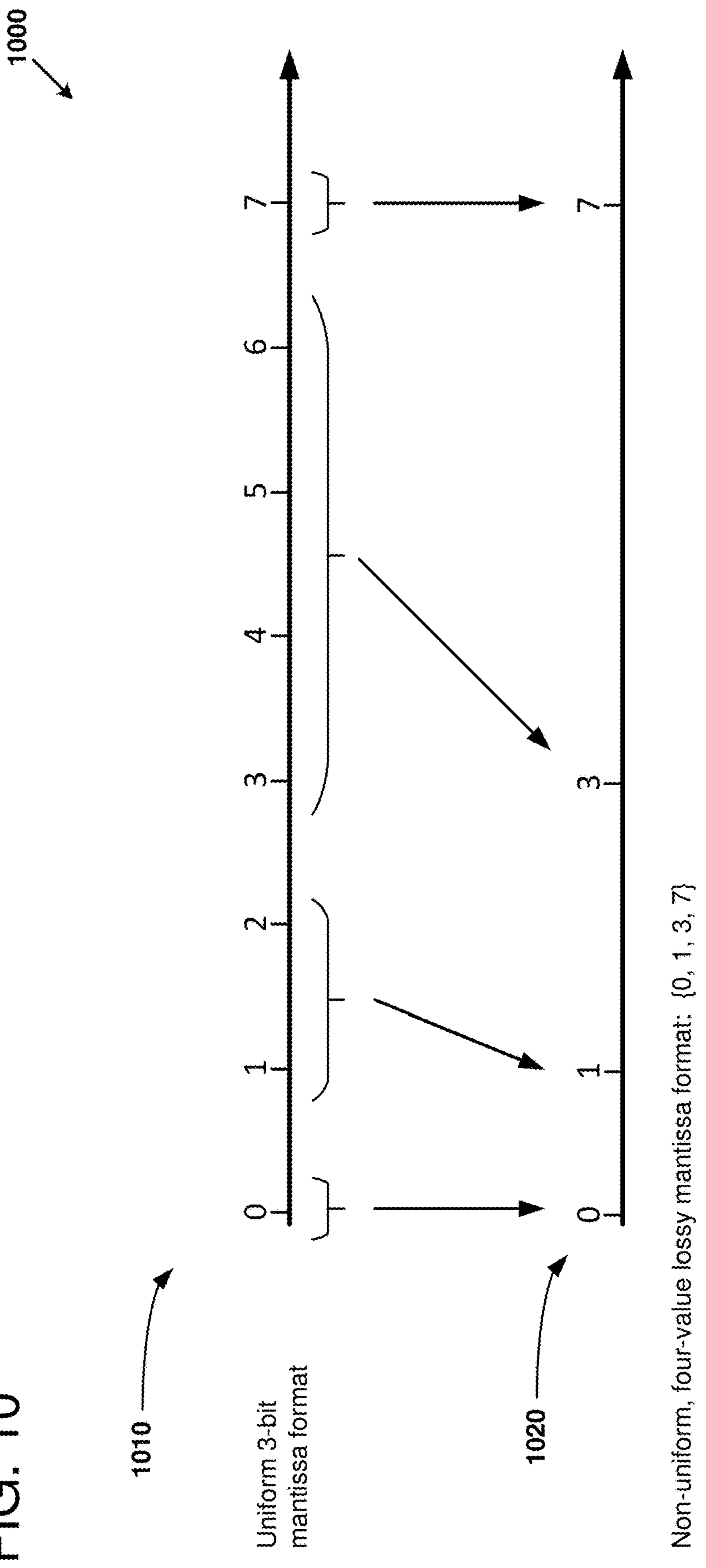

900
R(D)
2.5
2
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
920
910
$\frac{D}{\sigma^2}$

FIG. 10


1000
1010
Uniform 3-bit mantissa format
0
1
2
3
4
5
6
7
1020
Non-uniform, four-value lossy mantissa format: {0, 1, 3, 7}
0
1
3
7

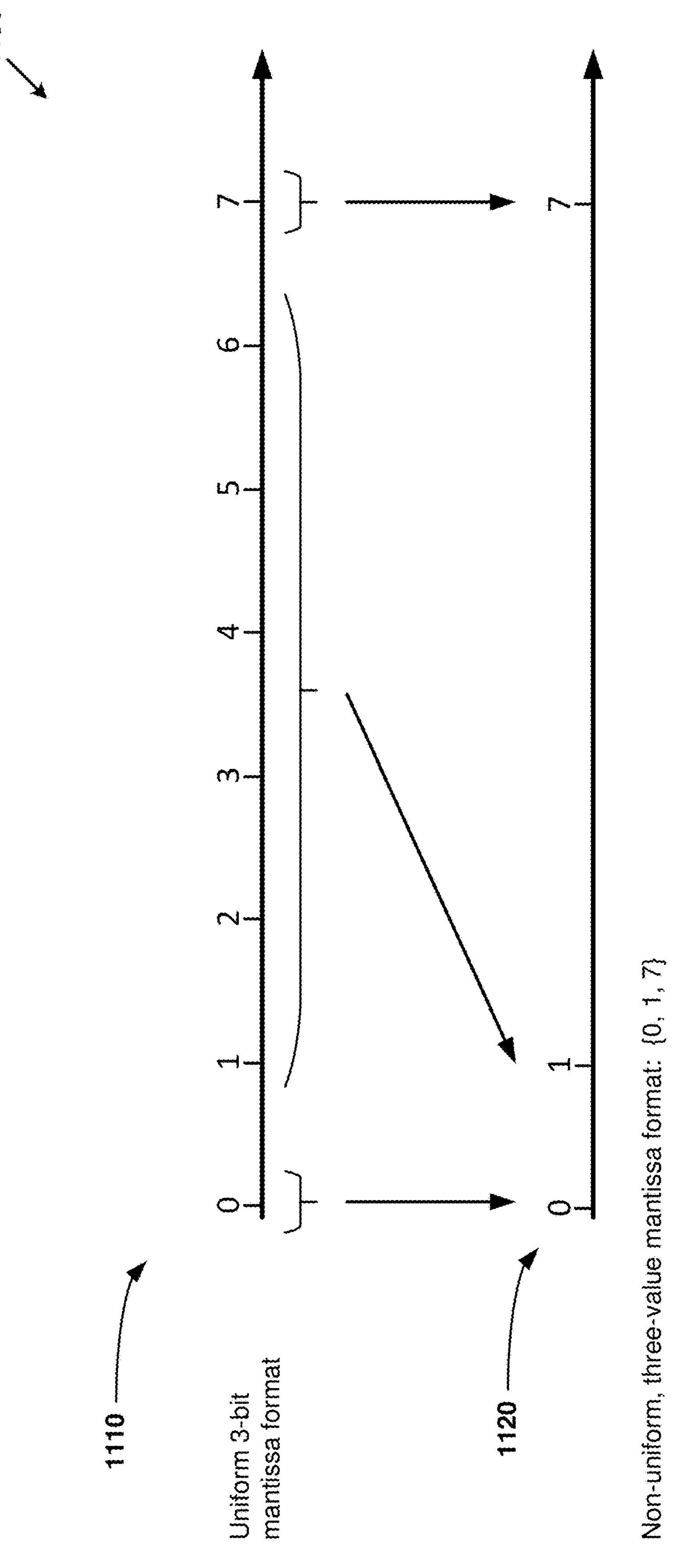

1100
1110
Uniform 3-bit mantissa format
0
1
2
3
4
5
6
7
1120
Non-uniform, three-value mantissa format: {0, 1, 7}
0
1
7

FIG. 11

FIG. 12

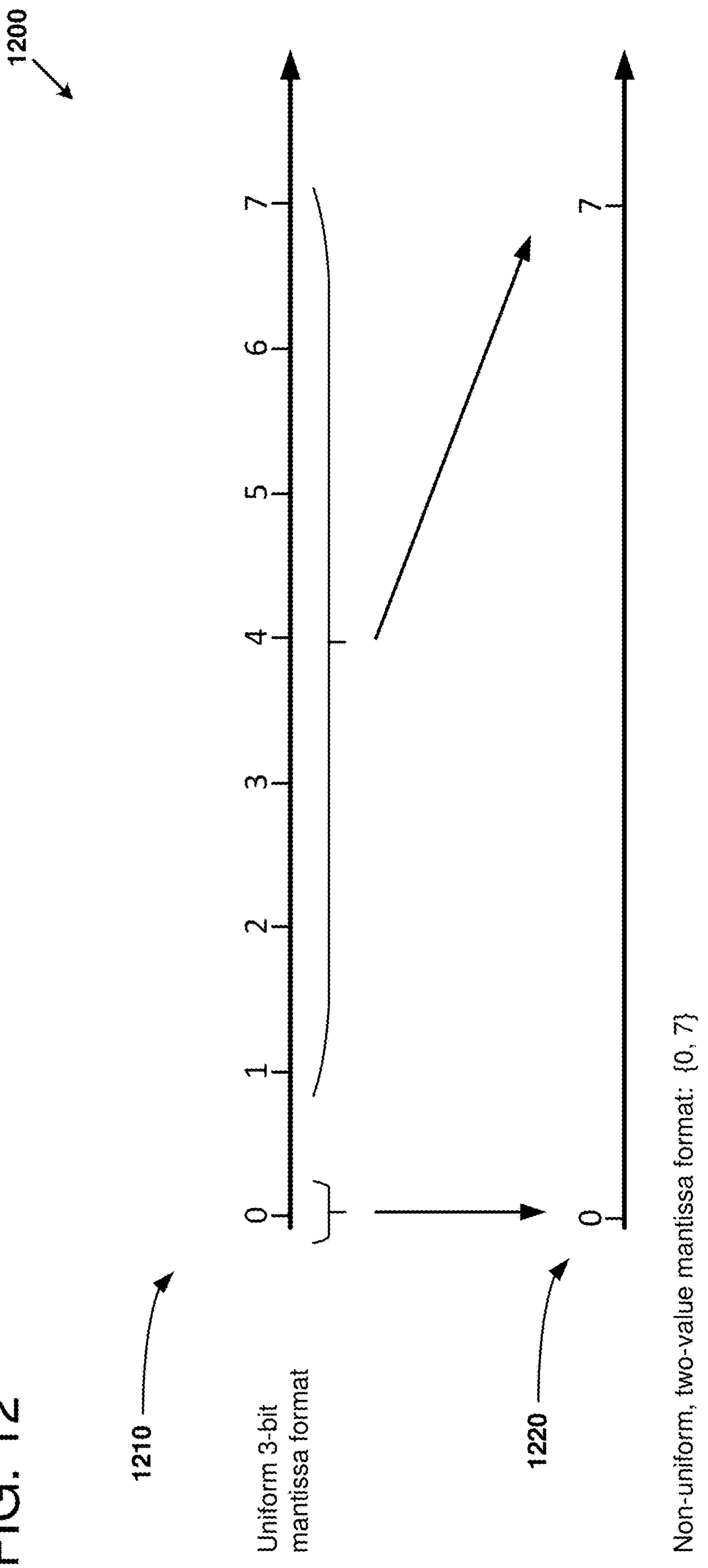

1200
1210
Uniform 3-bit mantissa format
0
1
2
3
4
5
6
7
1220
Non-uniform, two-value mantissa format: {0, 7}
0
7

FIG. 13

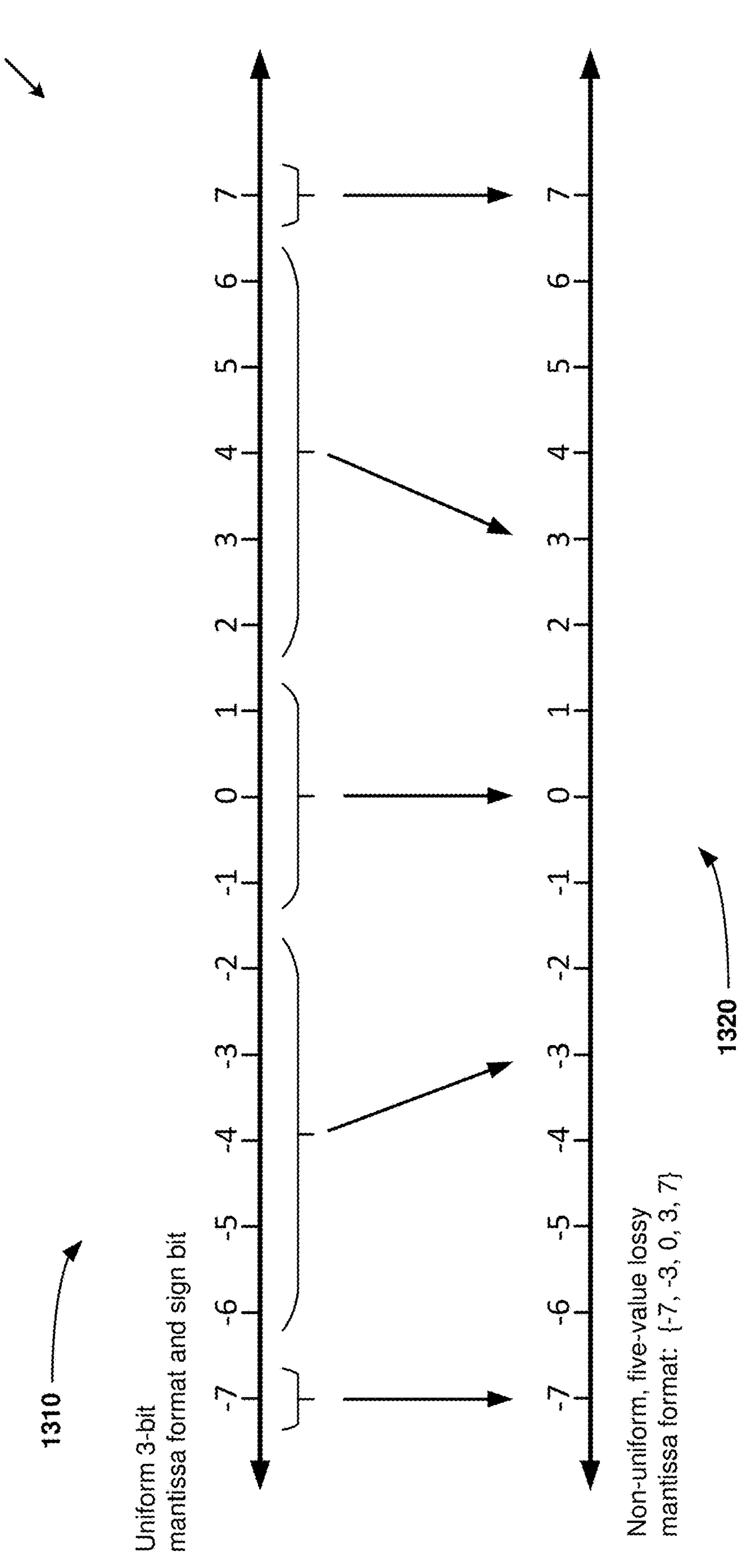

1300
1310
Uniform 3-bit mantissa format and sign bit
-7
-6
-5
-4
-3
-2
-1
0
1
2
3
4
5
6
7
1320
Non-uniform, five-value lossy mantissa format: {-7, -3, 0, 3, 7}

FIG. 14

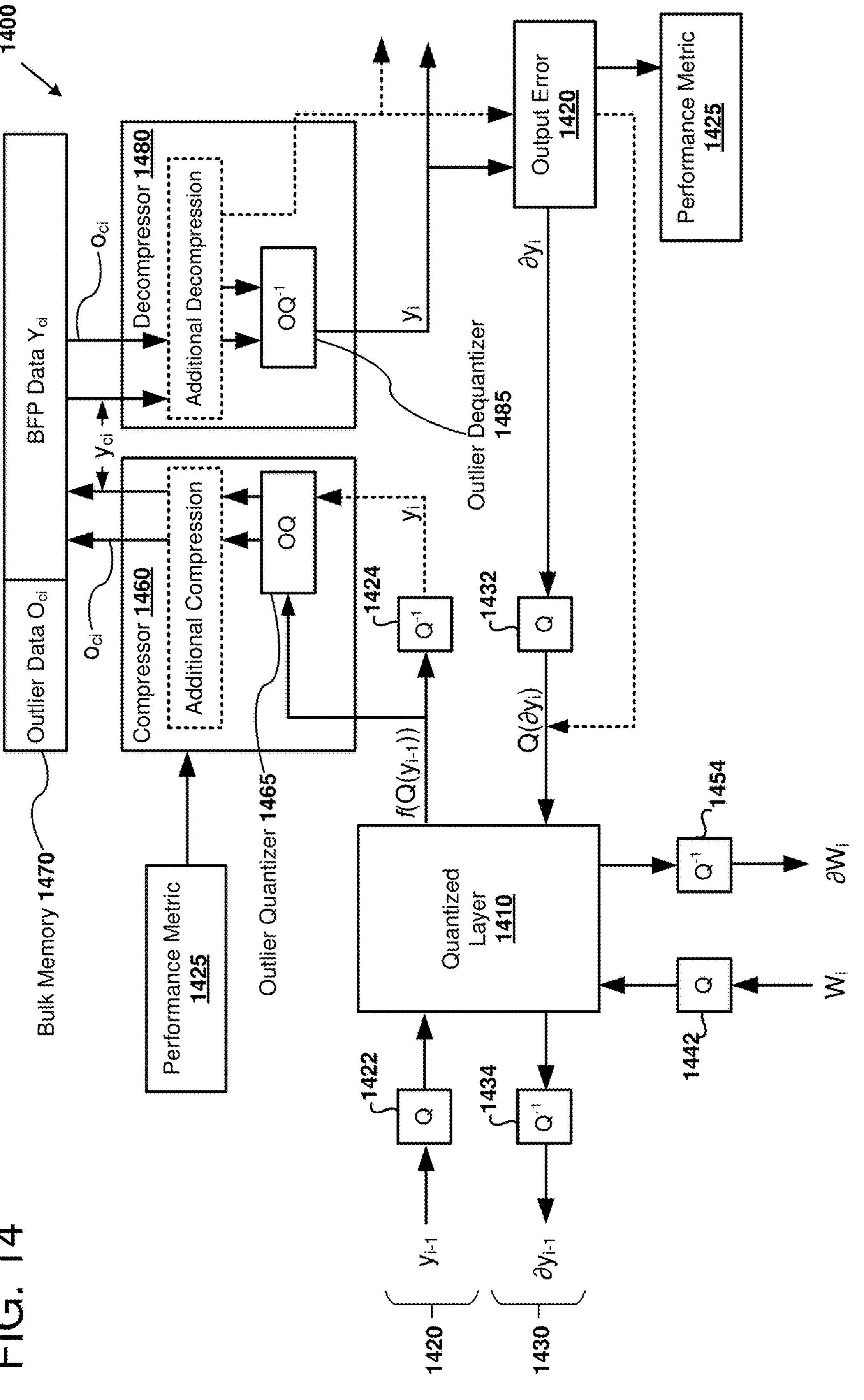
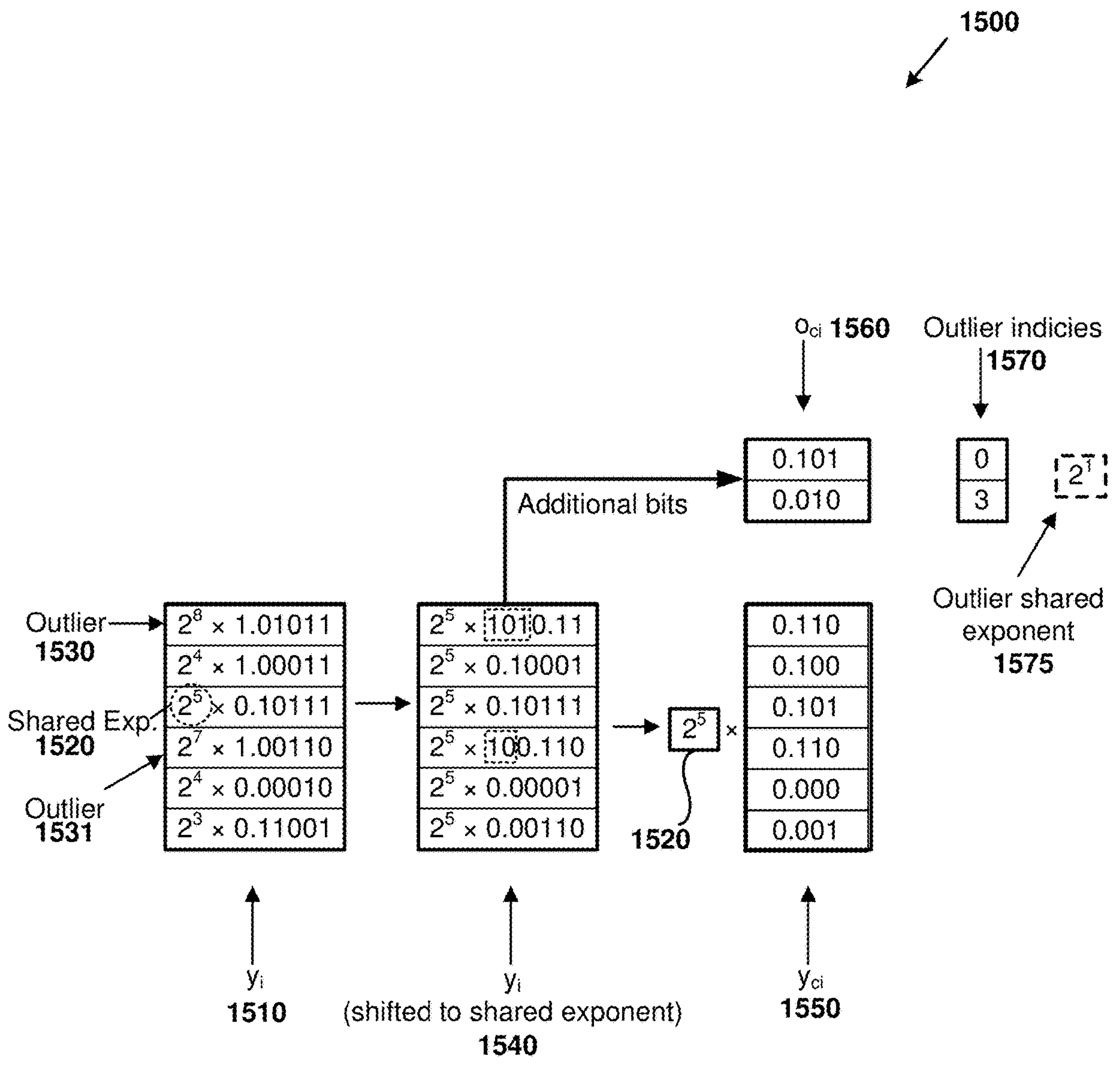

1400
Bulk Memory 1470
Outlier Data O_{ci}
BFP Data Y_{ci}
o_{ci}
y_{ci}
Compressor 1460
Additional Compression
OQ
Decompressor 1480
Additional Decompression
OQ^{-1}
Outlier Quantizer 1465
Outlier Dequantizer 1485
Performance Metric 1425
y_i
Output Error 1420
Performance Metric 1425
∂y_i
1424
Q^{-1}
1432
Q
$f(Q(y_{i-1}))$
$Q(\partial y_i)$
Quantized Layer 1410
1454
Q^{-1}
∂W_i
1442
Q
W_i
1422
Q
1434
Q^{-1}
y_{i-1}
∂y_{i-1}
1420
1430

FIG. 15


1500
o_{ci} 1560
Outlier indicies
1570
0.101
0.010
0
3
2^1
Additional bits
Outlier shared exponent
1575
Outlier
1530
Shared Exp.
1520
Outlier
1531
$2^8 \times 1.01011$
$2^4 \times 1.00011$
$2^5 \times 0.10111$
$2^7 \times 1.00110$
$2^4 \times 0.00010$
$2^3 \times 0.11001$
$2^5 \times 1010.11$
$2^5 \times 0.10001$
$2^5 \times 0.10111$
$2^5 \times 100.110$
$2^5 \times 0.00001$
$2^5 \times 0.00110$
2^5 ×
1520
0.110
0.100
0.101
0.110
0.000
0.001
y_i
1510
y_i
(shifted to shared exponent)
1540
y_{ci}
1550

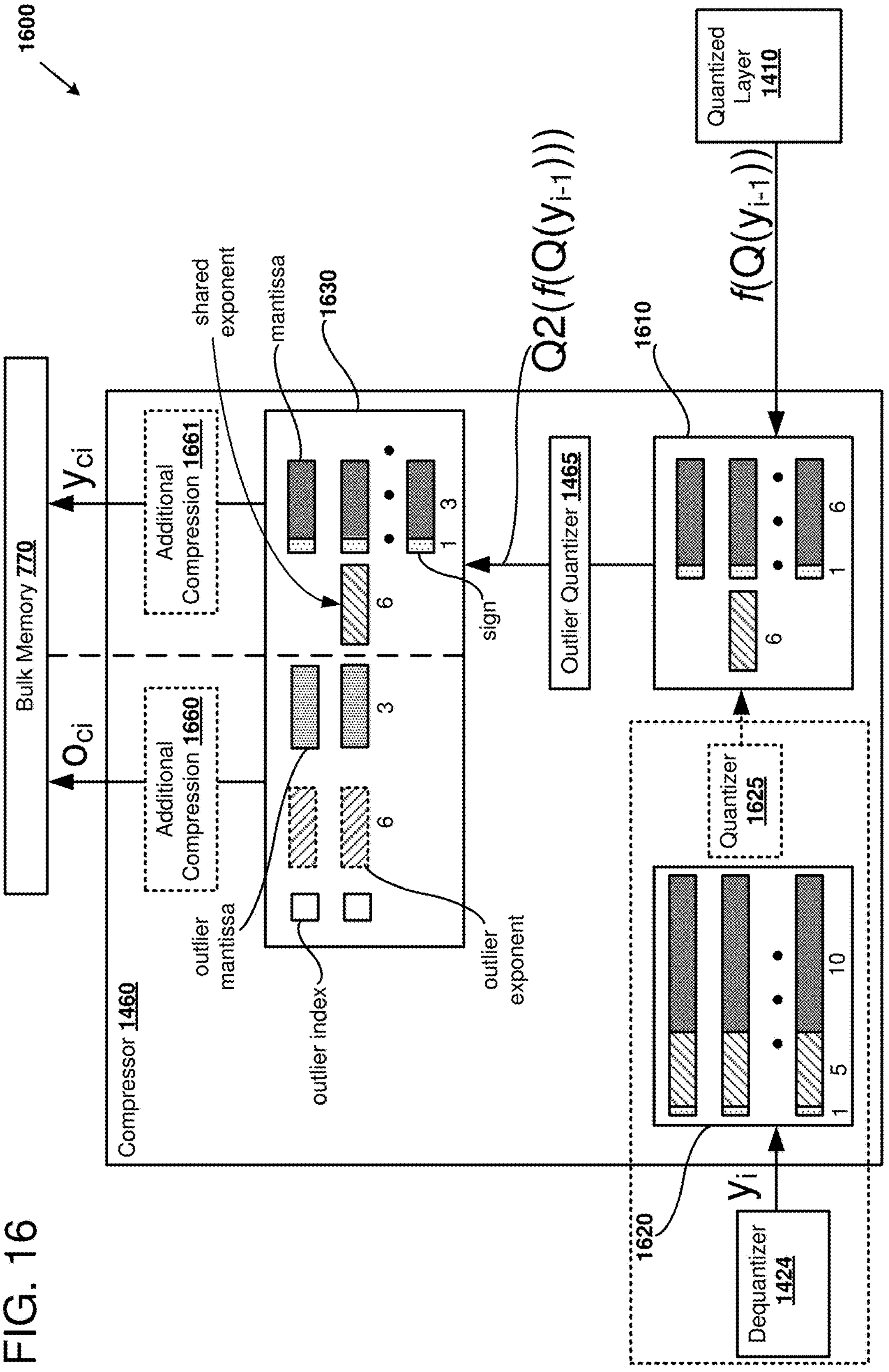

1600
Quantized Layer 1410
$f(Q(y_{i-1}))$
$Q2(f(Q(y_{i-1})))$
1610
6
1
6
Outlier Quantizer 1465
1630
shared exponent
mantissa
sign
6
1
3
Bulk Memory 770
y_{ci}
O_{ci}
Additional Compression 1661
Additional Compression 1660
6
3
outlier mantissa
outlier index
outlier exponent
Compressor 1460
Quantizer 1625
1620
1
5
10
y_i
Dequantizer 1424

FIG. 16

FIG. 17

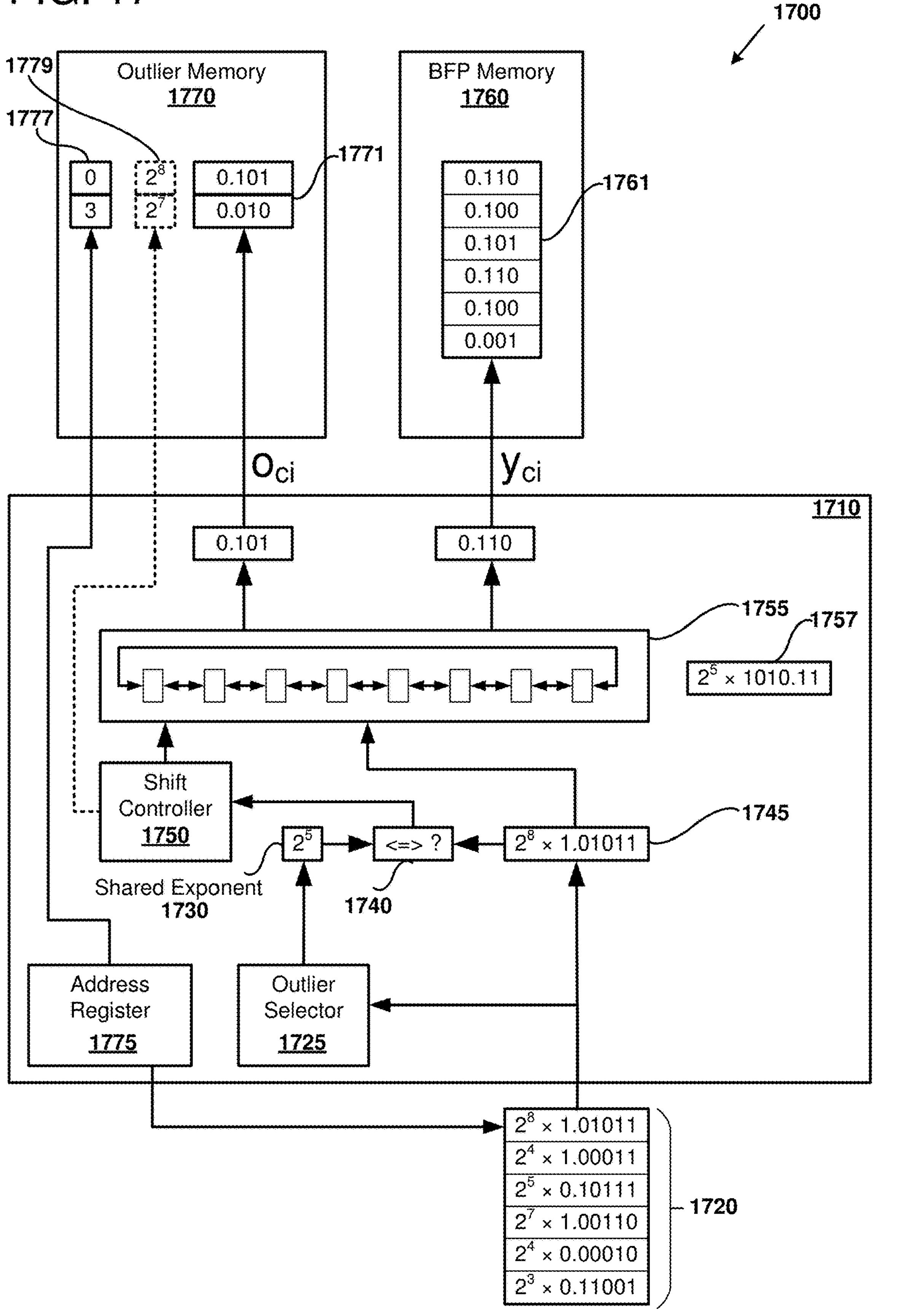

1700
Outlier Memory
1770
1779
1777
0
3
2^8
2^7
0.101
0.010
1771
BFP Memory
1760
0.110
0.100
0.101
0.110
0.100
0.001
1761
o_{ci}
y_{ci}
1710
0.101
0.110
1755
1757
$2^5 \times 1010.11$
Shift Controller
1750
2^5
<=> ?
$2^8 \times 1.01011$
1745
Shared Exponent
1730
1740
Address Register
1775
Outlier Selector
1725
$2^8 \times 1.01011$
$2^4 \times 1.00011$
$2^5 \times 0.10111$
$2^7 \times 1.00110$
$2^4 \times 0.00010$
$2^3 \times 0.11001$
1720

FIG. 18

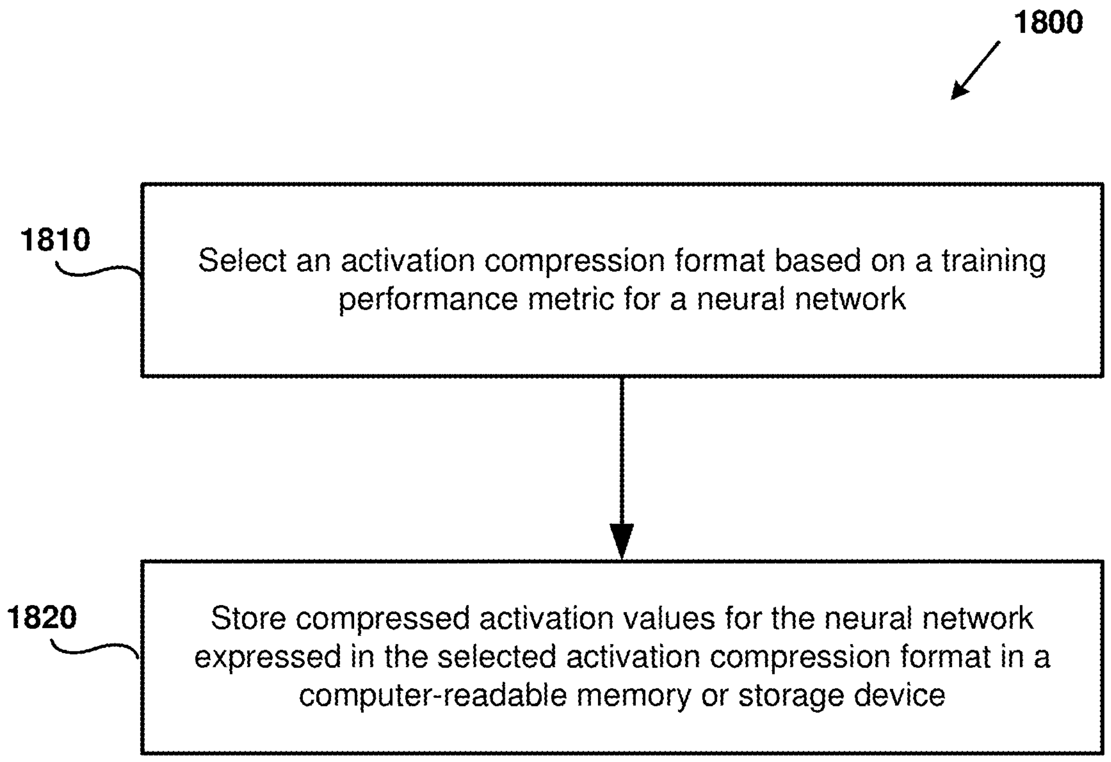

1800
1810
Select an activation compression format based on a training performance metric for a neural network
1820
Store compressed activation values for the neural network expressed in the selected activation compression format in a computer-readable memory or storage device

FIG. 19

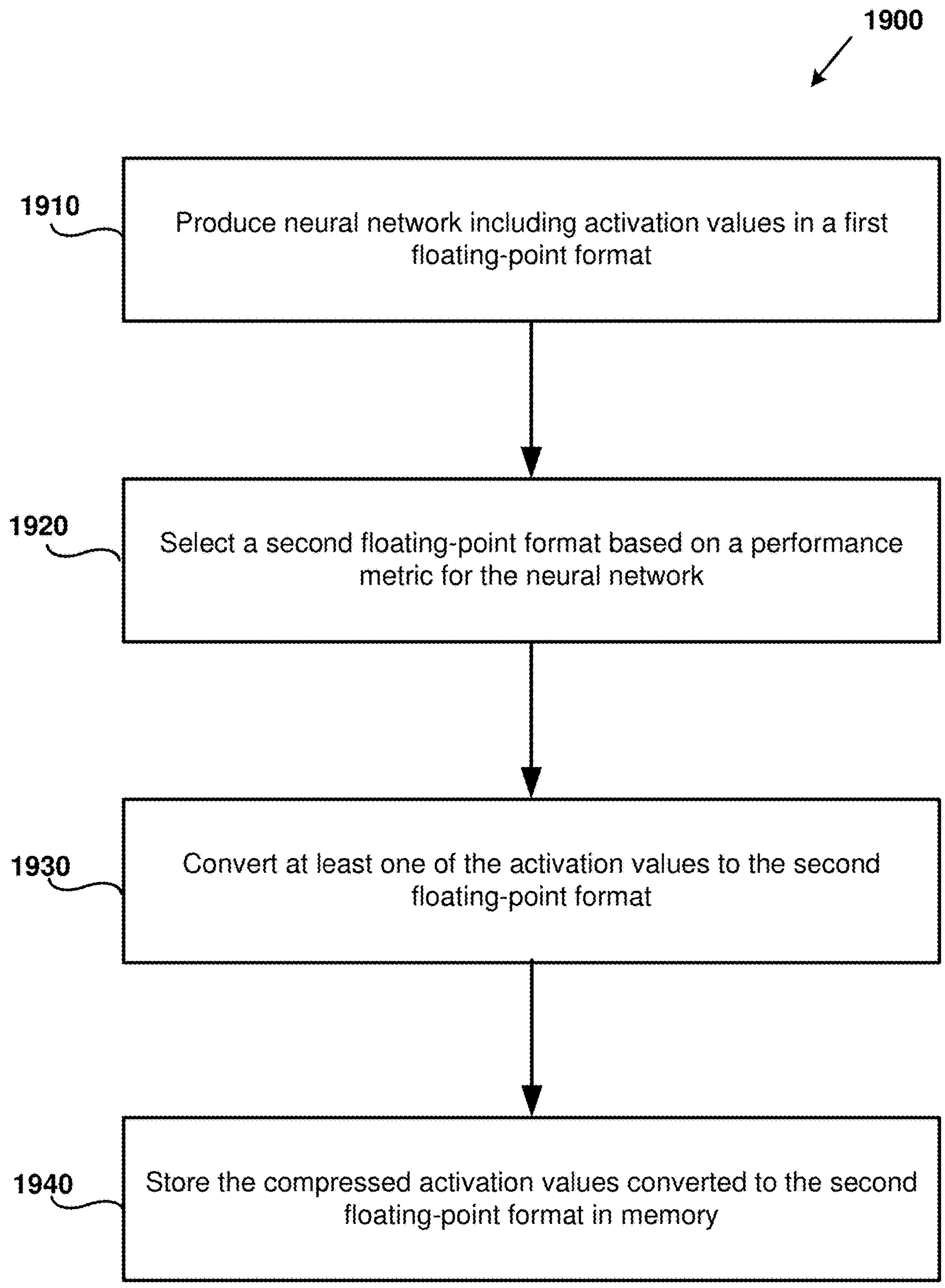

1900
1910
Produce neural network including activation values in a first floating-point format
1920
Select a second floating-point format based on a performance metric for the neural network
1930
Convert at least one of the activation values to the second floating-point format
1940
Store the compressed activation values converted to the second floating-point format in memory

FIG. 20

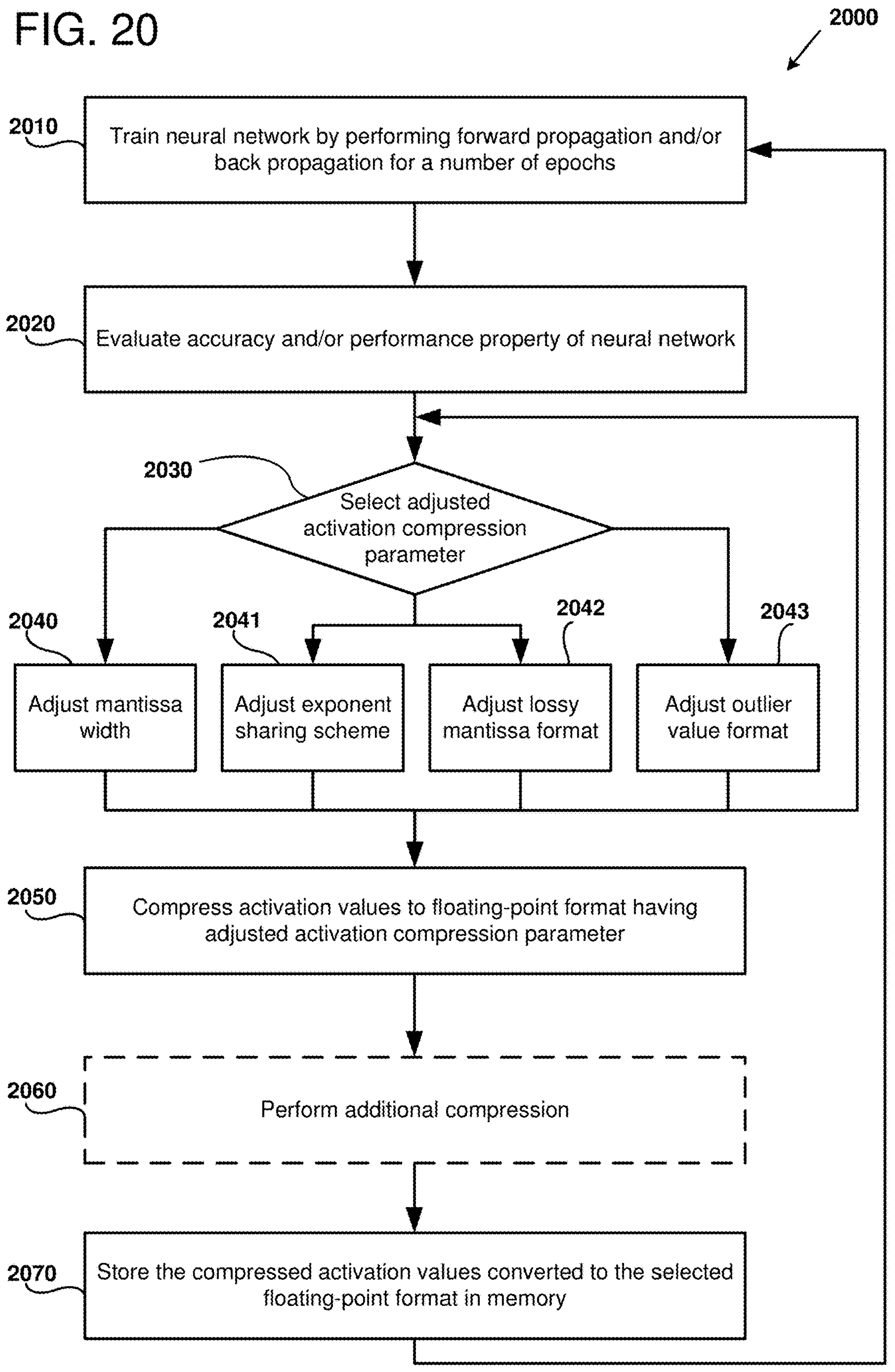
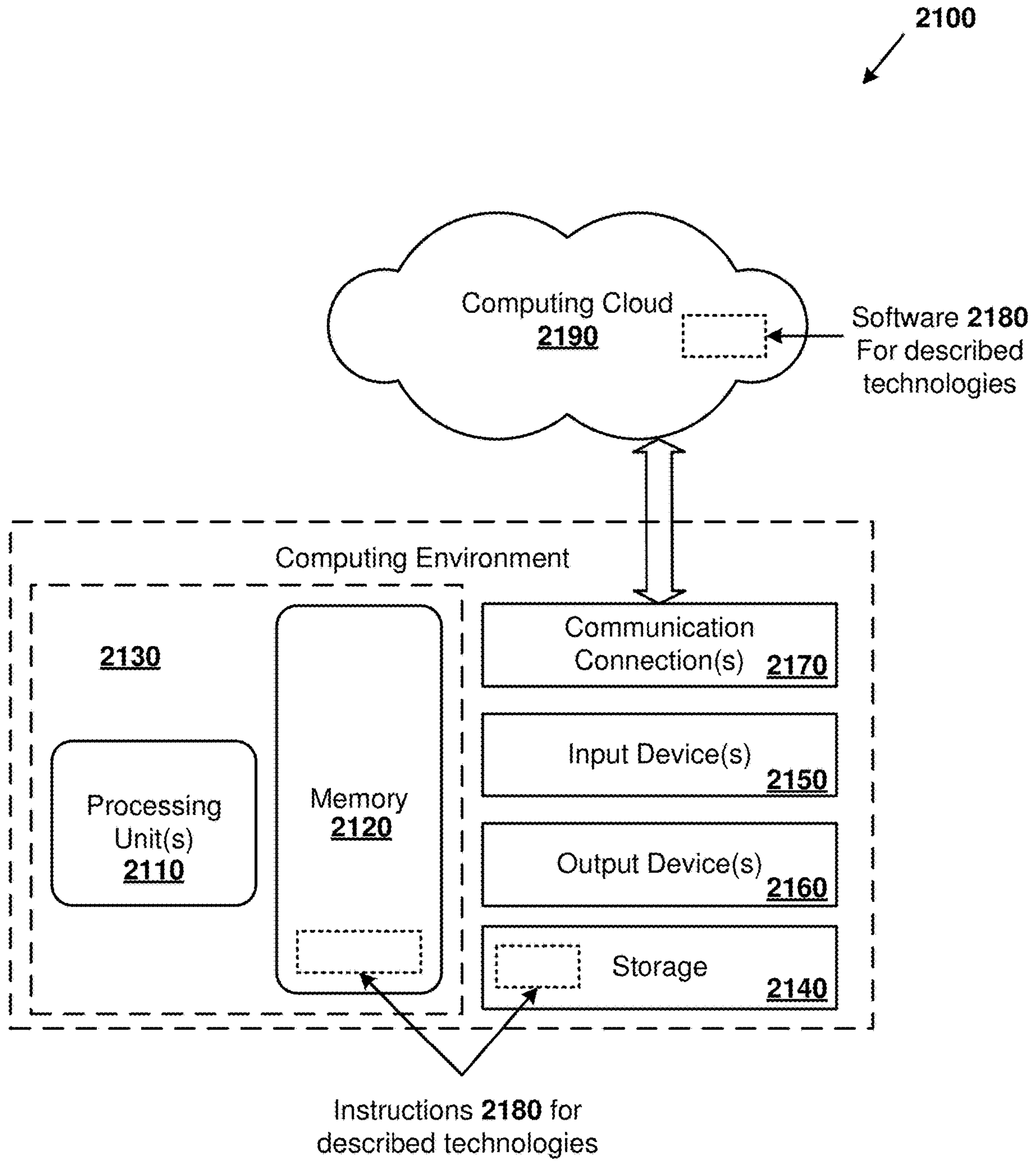

2000
2010
Train neural network by performing forward propagation and/or back propagation for a number of epochs
2020
Evaluate accuracy and/or performance property of neural network
2030
Select adjusted activation compression parameter
2040
Adjust mantissa width
2041
Adjust exponent sharing scheme
2042
Adjust lossy mantissa format
2043
Adjust outlier value format
2050
Compress activation values to floating-point format having adjusted activation compression parameter
2060
Perform additional compression
2070
Store the compressed activation values converted to the selected floating-point format in memory

FIG. 21


2100
Computing Cloud
2190
Software 2180
For described technologies
Computing Environment
2130
Processing Unit(s)
2110
Memory
2120
Communication Connection(s) 2170
Input Device(s) 2150
Output Device(s) 2160
Storage 2140
Instructions 2180 for described technologies

ADJUSTING ACTIVATION COMPRESSION FOR NEURAL NETWORK TRAINING

BACKGROUND

Machine learning (ML) and artificial intelligence (AI) techniques can be useful for solving a number of complex computational problems such as recognizing images and speech, analyzing and classifying information, and performing various classification tasks. Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to extract higher-level features from a set of training data. Specifically, the features can be extracted by training a model such as an artificial neural network (NN) or a deep neural network (DNN). After the model is trained, new data can be applied to the model and the new data can be classified (e.g., higher-level features can be extracted) using the trained model. Machine learning models are typically executed on a general-purpose processor (also referred to as a central processing unit (CPU)). However, training the models and/or using the models can be computationally expensive and so it may not be possible to use such technologies in real-time using general-purpose processors. Accordingly, there is ample opportunity for improvements in computer hardware and software to implement neural networks.

SUMMARY

Apparatus and methods are disclosed for storing activation values from a neural network in a compressed format for use during forward and backward propagation training of the neural network. Computing systems suitable for employing such neural networks include computers having general-purpose processors, neural network accelerators, or reconfigurable logic devices, such as Field Programmable Gate Arrays (FPGA). Activation values generated during forward propagation can be “stashed” (temporarily stored in bulk memory) in a compressed format and retrieved for use during backward propagation. The activation values used during training can be expressed in a normal precision, or a quantized or block floating-point format (BFP). The activation values stashed can be expressed in a further compressed format than the format used during the training. In some examples, the compressed formats include lossy or non-uniform mantissas for compressed values. In some examples, the compressed formats include storing outlier values for some but not all mantissa values. As training progresses, parameters of the compressed format can be adjusted, for example by increasing precision of the compressed format used to store activation values.

In some examples of the disclosed technology, a computer system includes general-purpose and/or special-purpose neural network processors, bulk memory including computer-readable storage devices or memory, and a floating-point compressor in communication with the bulk memory. As forward propagation occurs during training of neural network, activation values are produced in a first block floating-point format. The block floating-point is used to convert the activation values to a number format having a numerical precision less than the precision of the first block floating format. The compressed activation values are stored in the bulk memory for use during backward propagation.

This Summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used to limit the scope of the claimed subject matter. The foregoing and other objects, features, and advantages of the disclosed subject matter will become more apparent from the following detailed description, which proceeds with reference to the accompanying figures.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a quantization-enabled system for performing activation compression, as can be implemented in certain examples of the disclosed technology.

FIG. **2** is a diagram depicting an example of a deep neural network, as can be modeled using certain example methods and apparatus disclosed herein.

FIG. **3** is a diagram depicting certain aspects of converting a normal floating-point format to a quantized floating-point format, as can be performed in certain examples of the disclosed technology.

FIG. **4** depicts a number of example block floating-point formats that can be used to represent quantized neural network models, as can be used in certain examples of the disclosed technology.

FIG. **5** depicts a number of example block floating-point formats that can be used to represent quantized neural network models, as can be used in certain examples of the disclosed technology.

FIG. **6** is a flow chart depicting an example method of training a neural network for use with a quantized model, as can be implemented in certain examples of the disclosed technology.

FIG. **7** is a block diagram depicting an example environment for implementing activation compression, as can be implemented in certain examples of the disclosed technology.

FIG. **8** is block diagram depicting a further detailed example of activation compression, as can be implemented in certain examples of the disclosed technology.

FIG. **9** as a chart illustrating an example of the performance metric that can be used in selecting an adjusted precision parameter for compressed activation values, as can be used in certain examples of the disclosed technology.

FIG. **10** is a diagram illustrating converting a uniform three-bit mantissa format to a non-uniform, four-value lossy mantissa format, as can be implemented in certain examples of the disclosed technology.

FIG. **11** is a diagram illustrating converting a uniform three-bit mantissa format to a non-uniform, three-value lossy mantissa format, as can be implement it in certain examples of the disclosed technology.

FIG. **12** is a diagram illustrating converting a uniform three-bit mantissa format to a non-uniform, two-value the lossy mantissa format, as can be implemented in certain examples of the disclosed technology.

FIG. **13** is a diagram illustrating converting a sign value and three-bit mantissa format to a non-uniform five-value lossy mantissa format, as can be implement it in certain examples of the disclosed technology.

FIG. **14** is a block diagram depicting an example environment for implementing activation compression using outlier values, as can be implemented in certain examples of the disclosed technology.

FIG. **15** is a diagram illustrating an example a converting activation values to a second block floating-point format having outlier values, as can be implemented in certain examples of the disclosed technology.

FIG. **16** is block diagram depicting a further detailed example of activation compression using outlier values, as can be implemented in certain examples of the disclosed technology.

FIG. **17** is a block diagram depicting an example outlier quantizer, as can be implemented in certain examples of the disclosed technology.

FIG. **18** as a flowchart outlining an example method of storing compressed activation values, as can be performed in certain examples the disclosed technology.

FIG. **19** as a flowchart outlining an example method of storing compressed activation values in a selected floating-point format, as can be performed in certain examples of the disclosed technology.

FIG. **20** is a flowchart outlining an example method of adjusting activation compression parameters while training a neural network, as can be performed in certain examples of the disclosed technology.

FIG. **21** is a block diagram illustrating a suitable computing environment for implementing certain examples of the disclosed technology.

DETAILED DESCRIPTION

I. General Considerations

This disclosure is set forth in the context of representative embodiments that are not intended to be limiting in any way.

As used in this application the singular forms "a," "an," and "the" include the plural forms unless the context clearly dictates otherwise. Additionally, the term "includes" means "comprises." Further, the term "coupled" encompasses mechanical, electrical, magnetic, optical, as well as other practical ways of coupling or linking items together, and does not exclude the presence of intermediate elements between the coupled items. Furthermore, as used herein, the term "and/or" means any one item or combination of items in the phrase.

The systems, methods, and apparatus described herein should not be construed as being limiting in any way. Instead, this disclosure is directed toward all novel and non-obvious features and aspects of the various disclosed embodiments, alone and in various combinations and subcombinations with one another. The disclosed systems, methods, and apparatus are not limited to any specific aspect or feature or combinations thereof, nor do the disclosed things and methods require that any one or more specific advantages be present or problems be solved. Furthermore, any features or aspects of the disclosed embodiments can be used in various combinations and subcombinations with one another.

Although the operations of some of the disclosed methods are described in a particular, sequential order for convenient presentation, it should be understood that this manner of description encompasses rearrangement, unless a particular ordering is required by specific language set forth below. For example, operations described sequentially may in some cases be rearranged or performed concurrently. Moreover, for the sake of simplicity, the attached figures may not show the various ways in which the disclosed things and methods can be used in conjunction with other things and methods. Additionally, the description sometimes uses terms like "produce," "generate," "display," "receive," "verify," "execute," "perform," "convert," and "initiate" to describe the disclosed methods. These terms are high-level descriptions of the actual operations that are performed. The actual operations that correspond to these terms will vary depend-

ing on the particular implementation and are readily discernible by one of ordinary skill in the art.

Theories of operation, scientific principles, or other theoretical descriptions presented herein in reference to the apparatus or methods of this disclosure have been provided for the purposes of better understanding and are not intended to be limiting in scope. The apparatus and methods in the appended claims are not limited to those apparatus and methods that function in the manner described by such theories of operation.

Any of the disclosed methods can be implemented as computer-executable instructions stored on one or more computer-readable media (e.g., computer-readable media, such as one or more optical media discs, volatile memory components (such as DRAM or SRAM), or nonvolatile memory components (such as hard drives)) and executed on a computer (e.g., any commercially available computer, including smart phones or other mobile devices that include computing hardware). Any of the computer-executable instructions for implementing the disclosed techniques, as well as any data created and used during implementation of the disclosed embodiments, can be stored on one or more computer-readable media (e.g., computer-readable storage media). The computer-executable instructions can be part of, for example, a dedicated software application or a software application that is accessed or downloaded via a web browser or other software application (such as a remote computing application). Such software can be executed, for example, on a single local computer or in a network environment (e.g., via the Internet, a wide-area network, a local-area network, a client-server network (such as a cloud computing network), or other such network) using one or more network computers.

For clarity, only certain selected aspects of the software-based implementations are described. Other details that are well known in the art are omitted. For example, it should be understood that the disclosed technology is not limited to any specific computer language or program. For instance, the disclosed technology can be implemented by software written in C, C++, Java, or any other suitable programming language. Certain details of suitable computers and hardware are well-known and need not be set forth in detail in this disclosure.

Furthermore, any of the software-based embodiments (comprising, for example, computer-executable instructions for causing a computer to perform any of the disclosed methods) can be uploaded, downloaded, or remotely accessed through a suitable communication means. Such suitable communication means include, for example, the Internet, the World Wide Web, an intranet, software applications, cable (including fiber optic cable), magnetic communications, electromagnetic communications (including RF, microwave, and infrared communications), electronic communications, or other such communication means.

II. Overview of Quantized Artificial Neural Networks

Artificial Neural Networks (ANNs or as used throughout herein, "NNs") are applied to a number of applications in Artificial Intelligence and Machine Learning including image recognition, speech recognition, search engines, and other suitable applications. The processing for these applications may take place on individual devices such as personal computers or cell phones, but it may also be performed in large datacenters. At the same time, hardware accelerators that can be used with NNs include specialized NN processing units, such as tensor processing units (TPUs) and Field Programmable Gate Arrays (FPGAs) programmed to accelerate neural network processing. Such hardware devices are being deployed in consumer devices as well as in data centers due to their flexible nature and low power consumption per unit computation.

Traditionally NNs have been trained and deployed using single-precision floating-point (32-bit floating-point or float32 format). However, it has been shown that lower precision floating-point formats, such as 16-bit floating-point (float16) or fixed-point formats can be used to perform inference operations with minimal loss in accuracy. On specialized hardware, such as FPGAs, reduced precision formats can greatly improve the latency and throughput of DNN processing.

Numbers represented in normal-precision floating-point format (e.g., a floating-point number expresses in a 16-bit floating-point format, a 32-bit floating-point format, a 64-bit floating-point format, or an 80-bit floating-point format) can be converted to quantized-precision format numbers may allow for performance benefits in performing operations. In particular, NN weights and activation values can be represented in a lower-precision quantized format with an acceptable level of error introduced. Examples of lower-precision quantized formats include formats having a reduced bit width (including by reducing the number of bits used to represent a number's mantissa or exponent) and block floating-point formats where two or more numbers share the same single exponent.

One of the characteristics of computation on an FPGA device is that it typically lacks hardware floating-point support. Floating-point operations may be performed at a penalty using the flexible logic, but often the amount of logic needed to support floating-point is prohibitive in FPGA implementations. Some newer FPGAs have been developed that do support floating-point computation, but even on these the same device can produce twice as many computational outputs per unit time as when it is used in an integer mode. Typically, NNs are created with floating-point computation in mind, but when an FPGA is targeted for NN processing it would be beneficial if the neural network could be expressed using integer arithmetic. Examples of the disclosed technology include hardware implementations of block floating-point (BFP), including the use of BFP in NN, FPGA, and other hardware environments.

A typical floating-point representation in a computer system consists of three parts: sign (s), exponent (e), and mantissa (m). The sign indicates if the number is positive or negative. The exponent and mantissa are used as in scientific notation:

$$\text{Value}=s\times m\times 2^{e} \quad \text{(Eq. 1)}$$

Any number may be represented, within the precision limits of the mantissa. Since the exponent scales the mantissa by powers of 2, just as the exponent does by powers of 10 in scientific notation, the magnitudes of very large numbers may be represented. The precision of the representation is determined by the precision of the mantissa. Typical floating-point representations use a mantissa of 10 (float 16), 24 (float 32), or 53 (float64) bits in width. An integer with magnitude greater than 2^{53} can be approximated in a float64 floating-point format, but it will not be represented exactly because there are not enough bits in the mantissa. A similar effect can occur for arbitrary fractions where the fraction is represented by bits of the mantissa that take on the value of negative powers of 2. There are many fractions that cannot be exactly represented because they are irrational in a binary number system. More exact representations are possible in both situations, but they may require the mantissa to contain more bits. Ultimately, an infinite number of mantissa bits are

required to represent some numbers exactly (e.g., $1/3=0.\overline{3}$; $22/7=3.\overline{142857}$). The 10-bit (half precision float), 24-bit (single precision float), and 53-bit (double precision float) mantissa limits are common compromises of mantissa storage requirements versus representation precision in general-purpose computers.

With block floating-point formats, a group of two or more numbers use a single shared exponent with each number still having its own sign and mantissa. In some examples, the shared exponent is chosen to be the largest exponent of the original floating-point values. For purposes of the present disclosure, the term block floating-point (BFP) means a number system in which a single exponent is shared across two or more values, each of which is represented by a sign and mantissa pair (whether there is an explicit sign bit, or the mantissa itself is signed). In some examples, all values of one or more rows or columns of a matrix or vector, or all values of a matrix or vector, can share a common exponent. In other examples, the BFP representation may be unsigned. In some examples, some but not all of the elements in a matrix or vector BFP representation may include numbers represented as integers, floating-point numbers, fixed point numbers, symbols, or other data formats mixed with numbers represented with a sign, mantissa, and exponent. In some examples, some or all of the elements in a matrix or vector BFP representation can include complex elements having two or more parts, for example: complex numbers with an imaginary component ($a+bi$, where $i=\sqrt{-1}$); fractions including a numerator and denominator, in polar coordinates (r, θ), or other multi-component element.

Parameters of particular BFP formats can be selected for a particular implementation to tradeoff precision and storage requirements. For example, rather than storing an exponent with every floating-point number, a group of numbers can share the same exponent. To share exponents while maintaining a high level of accuracy, the numbers should have close to the same magnitude, since differences in magnitude are expressed in the mantissa. If the differences in magnitude are too great, the mantissa will overflow for the large values, or may be zero ("underflow") for the smaller values. Depending on a particular application, some amount of overflow and/or underflow may be acceptable.

The size of the mantissa can be adjusted to fit a particular application. This can affect the precision of the number being represented, but potential gains are realized from a reduced representation size. For example, a normal single-precision float has a size of four bytes, but for certain implementations of the disclosed technology, only two bytes are used to represent the sign and mantissa of each value. In some implementations, the sign and mantissa of each value can be represented in a byte or less.

In certain examples of the disclosed technology, the representation expressed above is used to derive the original number from the representation, but only a single exponent is stored for a group of numbers, each of which is represented by a signed mantissa. Each signed mantissa can be represented by two bytes or less, so in comparison to four-byte floating-point, the memory storage savings is about 2×. Further, the memory bandwidth requirements of loading and storing these values are also approximately one-half that of normal floating-point.

Neural network operations are used in many artificial intelligence operations. Often, the bulk of the processing operations performed in implementing a neural network is in performing Matrix×Matrix or Matrix×Vector multiplications or convolution operations. Such operations are compute- and memory-bandwidth intensive, where the size of a matrix may be, for example, 1000×1000 elements (e.g., 1000×1000 numbers, each including a sign, mantissa, and exponent) or larger and there are many matrices used. As discussed herein, BFP techniques can be applied to such operations to reduce the demands for computation as well as memory bandwidth in a given system, whether it is an FPGA, CPU, or another hardware platform. As used herein, the use of the term "element" herein refers to a member of such a matrix or vector.

As used herein, the term "tensor" refers to a multi-dimensional array that can be used to represent properties of a NN and includes one-dimensional vectors as well as two-, three-, four-, or larger dimension matrices. As used in this disclosure, tensors do not require any other mathematical properties unless specifically stated.

As used herein, the term "normal-precision floating-point" refers to a floating-point number format having a mantissa, exponent, and optionally a sign and which is natively supported by a native or virtual CPU. Examples of normal-precision floating-point formats include, but are not limited to, IEEE 754 standard formats such as 16-bit, 32-bit, 64-bit, or to other processors supported by a processor, such as Intel AVX, AVX2, IA32, x86_64, or 80-bit floating-point formats.

As used herein, the term "lossy mantissa" refers to a mantissa that represents a higher-precision mantissa as a discrete set of lower-precision mantissa values. For example, for a four-bit number having a sign bit and three mantissa bits, the three-bit mantissa (which can represent 8 values, for example, the set {0, 1, 2, 3, 4, 5, 6, 7}, may be converted to a lossy mantissa having discrete sets of mantissa values, for example, any one of the sets: {0, 1, 3, 7}, {0, 1, 7}, or {0, 7}, depending on a selected lossy mantissa scheme. The underlying representation of the lossy mantissa may vary. For the preceding three sets, an example set of respective binary representations is {00, 01, 10, 11}; {00, 10, 11}; and {0, 1}; respectively.

A used herein, the term "non-uniform mantissa" refers to a property of certain lossy mantissa that at least one of the values being represented are not uniformly distributed. For the example in the preceding paragraph, the three-bit mantissa is uniform, as every value is one unit apart. The three sets of lossy mantissas are also non-uniform, being distributed differently: 1, 2, and 4 units apart for the first set; 1 or 6 units apart for the second set, and 7 units apart for the third set.

A given number can be represented using different precision (e.g., different quantized precision) formats. For example, a number can be represented in a higher precision format (e.g., float32) and a lower precision format (e.g., float16). Lowering the precision of a number can include reducing the number of bits used to represent the mantissa or exponent of the number. Additionally, lowering the precision of a number can include reducing the range of values that can be used to represent an exponent of the number, such as when multiple numbers share a common exponent. Similarly, increasing the precision of a number can include increasing the number of bits used to represent the mantissa or exponent of the number. Additionally, increasing the precision of a number can include increasing the range of values that can be used to represent an exponent of the number, such as when a number is separated from a group of numbers that shared a common exponent. As used herein, converting a number from a higher precision format to a lower precision format may be referred to as down-casting or quantizing the number. Converting a number from a lower precision format to a higher precision format may be referred to as up-casting or de-quantizing the number.

As used herein, the term "quantized-precision floating-point" refers to a floating-point number format where two or more values of a tensor have been modified to have a lower precision than when the values are represented in normal-precision floating-point. In particular, many examples of quantized-precision floating-point representations include block floating-point formats, where two or more values of the tensor are represented with reference to a common exponent. The quantized-precision floating-point number can be generated by selecting a common exponent for two, more, or all elements of a tensor and shifting mantissas of individual elements to match the shared, common exponent. In some examples, groups of elements within a tensor can share a common exponent on, for example, a per-row, per-column, per-tile, or other basis.

In one example of the disclosed technology, a neural network accelerator is configured to performing training operations for layers of a neural network, including forward propagation and back propagation. The values of one or more of the neural network layers can be expressed in a quantized format, that has lower precision than normal-precision floating-point formats. For example, block floating-point formats can be used to accelerate computations performed in training and inference operations using the neural network accelerator. Use of quantized formats can improve neural network processing by, for example, allowing for faster hardware, reduced memory overhead, simpler hardware design, reduced energy use, reduced integrated circuit area, cost savings and other technological improvements. It is often desirable that operations be performed to mitigate noise or other inaccuracies introduced by using lower-precision quantized formats. Further, portions of neural network training, such as temporary storage of activation values, can be improved by compressing a portion of these values (e.g., for an input, hidden, or output layer of a neural network), either from normal-precision floating-point or from a first block floating-point, to a lower precision number format having lossy or non-uniform mantissas. The activation values can be later retrieved and dequantized for use during, for example, back propagation during the training phase.

An input tensor for the given layer can be converted from a normal-precision floating-point format to a quantized-precision floating-point format. A tensor operation can be performed using the converted input tensor having lossy or non-uniform mantissas. A result of the tensor operation can be converted from the block floating-point format to the normal-precision floating-point format. The tensor operation can be performed during a forward-propagation mode or a back-propagation mode of the neural network. For example, during a back-propagation mode, the input tensor can be an output error term from a layer adjacent to (e.g., following) the given layer or weights of the given layer. As another example, during a forward-propagation mode, the input tensor can be an output term from a layer adjacent to (e.g., preceding) the given layer or weights of the given layer. The converted result can be used to generate an output tensor of the layer of the neural network, where the output tensor is in normal-precision floating-point format. In this manner, the neural network accelerator can potentially be made smaller and more efficient than a comparable accelerator that uses only a normal-precision floating-point format. A smaller and more efficient accelerator may have increased computational performance and/or increased energy efficiency. Additionally, the neural network accelerator can potentially have increased accuracy compared to an accelerator that uses only a quantized-precision floating-point format. By increasing the accuracy of the accelerator, a convergence time for training may be decreased and the accelerator may be more accurate when classifying inputs to the neural network. Reducing the computational complexity of using the models can potentially decrease the time to extract a feature during inference, decrease the time for adjustment during training, and/or reduce energy consumption during training and/or inference.

III. Example Architectures for Implementing Activation Compression with Non-Uniform Mantissa Block Floating-Point

FIG. **1** is a block diagram **100** outlining an example quantization-enabled system **110** as can be implemented in certain examples of the disclosed technology, including for use in compression of activation values during training of neural networks. As shown in FIG. **1**, the quantization-enabled system **110** can include a number of hardware resources including general-purpose processors **120** and special-purpose processors such as graphics processing units **122** and neural network accelerator **180**. The processors are coupled to memory **125** and storage **129**, which can include volatile or non-volatile memory devices. The processors **120** and **122** execute instructions stored in the memory or storage in order to provide a neural network module **130**. The neural network module **130** includes software interfaces that allow the system to be programmed to implement various types of neural networks. For example, software functions can be provided that allow applications to define neural networks including weights, biases, activation functions, node values, and interconnections between layers of a neural network. Additionally, software functions can be used to define state elements for recurrent neural networks. The neural network module **130** can further provide utilities to allow for training and retraining of a neural network implemented with the module. Values representing the neural network module are stored in memory or storage and are operated on by instructions executed by one of the processors. The values stored in memory or storage can be represented using normal-precision floating-point and/or quantized floating-point values, including floating-point values having lossy or non-uniform mantissas or outlier mantissas.

In some examples, proprietary or open source libraries or frameworks are provided to a programmer to implement neural network creation, training, and evaluation. Examples of such libraries include TensorFlow, Microsoft Cognitive Toolkit (CNTK), Caffe, Theano, and Keras. In some examples, programming tools such as integrated development environments provide support for programmers and users to define, compile, and evaluate NNs.

The neural network accelerator **180** can be implemented as a custom or application-specific integrated circuit (e.g., including a system-on-chip (SoC) integrated circuit), as a field programmable gate array (FPGA) or other reconfigurable logic, or as a soft processor virtual machine hosted by a physical, general-purpose processor. The neural network accelerator **180** can include a tensor processing unit **182**, reconfigurable logic devices **184**, and/or one or more neural processing cores (such as the quantization accelerator **186**). The quantization accelerator **186** can be configured in hardware, software, or a combination of hardware and software. As one example, the quantization accelerator **186** can be configured and/or executed using instructions executable on the tensor processing unit **182**. As another example, the quantization accelerator **186** can be configured by programming reconfigurable logic blocks **184**. As another example,

the quantization accelerator **186** can be configured using hard-wired logic gates of the neural network accelerator **180**.

The quantization accelerator **186** can be programmed to execute a subgraph, an individual layer, or a plurality of layers of a neural network. For example, the quantization accelerator **186** can be programmed to perform operations for all or a portion of a layer of a NN. The quantization accelerator **186** can access a local memory used for storing weights, biases, input values, output values, forget values, state values, and so forth. The quantization accelerator **186** can have many inputs, where each input can be weighted by a different weight value. For example, the quantization accelerator **186** can produce a dot product of an input tensor and the programmed input weights for the quantization accelerator **186**. In some examples, the dot product can be adjusted by a bias value before it is used as an input to an activation function. The output of the quantization accelerator **186** can be stored in the local memory, where the output value can be accessed and sent to a different NN processor core and/or to the neural network module **130** or the memory **125**, for example. Intermediate values in the quantization accelerator can often be stored in a smaller or more local memory, while values that may not be needed until later in a training process can be stored in a "bulk memory" a larger, less local memory (or storage device, such as on an SSD (solid state drive) or hard drive). For example, during training forward propagation, once activation values for a next layer in the NN have been calculated, those values may not be accessed until for propagation through all layers has completed. Such activation values can be stored in such a bulk memory.

The neural network accelerator **180** can include a plurality **110** of quantization accelerators **186** that are connected to each other via an interconnect (not shown). The interconnect can carry data and control signals between individual quantization accelerators **186**, a memory interface (not shown), and an input/output (I/O) interface (not shown). The interconnect can transmit and receive signals using electrical, optical, magnetic, or other suitable communication technology and can provide communication connections arranged according to a number of different topologies, depending on a particular desired configuration. For example, the interconnect can have a crossbar, a bus, a point-to-point bus, or other suitable topology. In some examples, any one of the plurality of quantization accelerators **186** can be connected to any of the other cores, while in other examples, some cores are only connected to a subset of the other cores. For example, each core may only be connected to a nearest 4, 8, or 10 neighboring cores. The interconnect can be used to transmit input/output data to and from the quantization accelerators **186**, as well as transmit control signals and other information signals to and from the quantization accelerators **186**. For example, each of the quantization accelerators **186** can receive and transmit semaphores that indicate the execution status of operations currently being performed by each of the respective quantization accelerators **186**. Further, matrix and vector values can be shared between quantization accelerators **186** via the interconnect. In some examples, the interconnect is implemented as wires connecting the quantization accelerators **186** and memory system, while in other examples, the core interconnect can include circuitry for multiplexing data signals on the interconnect wire(s), switch and/or routing components, including active signal drivers and repeaters, or other suitable circuitry. In some examples of the disclosed technology, signals transmitted within and to/from neural network accelerator **180** are not limited to full swing electrical digital signals, but the neural network accelerator **180** can be configured to include differential signals, pulsed signals, or other suitable signals for transmitting data and control signals.

In some examples, the quantization-enabled system **110** can include an optional quantization emulator that emulates functions of the neural network accelerator **180**. The neural network accelerator **180** provides functionality that can be used to convert data represented in full precision floating-point formats in the neural network module **130** into quantized format values. The neural network accelerator **180** can also perform operations using quantized format values. Such functionality will be discussed in further detail below.

The neural network module **130** can be used to specify, train, and evaluate a neural network model using a tool flow that includes a hardware-agnostic modelling framework **131** (also referred to as a native framework or a machine learning execution engine), a neural network compiler **132**, and a neural network runtime environment **133**. The memory includes computer-executable instructions for the tool flow including the modelling framework **131**, the neural network compiler **132**, and the neural network runtime environment **133**. The tool flow can be used to generate neural network data **200** representing all or a portion of the neural network model, such as the neural network model discussed below regarding FIG. **2**. It should be noted that while the tool flow is described as having three separate tools (**131**, **132**, and **133**), the tool flow can have fewer or more tools in various examples. For example, the functions of the different tools (**131**, **132**, and **133**) can be combined into a single modelling and execution environment. In other examples, where a neural network accelerator is deployed, such a modeling framework may not be included.

The neural network data **200** can be stored in the memory **125**, which can include local memory **126**, which is typically implemented as static read only memory (SRAM), embedded dynamic random access memory (eDRAM), in latches or flip-flops in a register file, in a block RAM, or other suitable structure, and bulk memory **127**, which is typically implemented in memory structures supporting larger, but often slower access than the local memory **126**. For example, the bulk memory may be off-chip DRAM, network accessible RAM, SSD drives, hard drives, or network-accessible storage. Depending on a particular memory technology available, other memory structures, including the foregoing structures recited for the local memory, may be used to implement bulk memory. The neural network data **200** can be represented in one or more formats. For example, the neural network data **200** corresponding to a given neural network model can have a different format associated with each respective tool of the tool flow. Generally, the neural network data **200** can include a description of nodes, edges, groupings, weights, biases, activation functions, and/or tensor values. As a specific example, the neural network data **200** can include source code, executable code, metadata, configuration data, data structures and/or files for representing the neural network model.

The modelling framework **131** can be used to define and use a neural network model. As one example, the modelling framework **131** can include pre-defined APIs and/or programming primitives that can be used to specify one or more aspects of the neural network model. The pre-defined APIs can include both lower-level APIs (e.g., activation functions, cost or error functions, nodes, edges, and tensors) and higher-level APIs (e.g., layers, convolutional neural networks, recurrent neural networks, linear classifiers, and so forth). "Source code" can be used as an input to the modelling framework **131** to define a topology of the graph of a given neural network model. In particular, APIs of the modelling framework **131** can be instantiated and interconnected within the source code to specify a complex neural network model. A data scientist can create different neural network models by using different APIs, different numbers of APIs, and interconnecting the APIs in different ways.

In addition to the source code, the memory **125** can also store training data. The training data includes a set of input data for applying to the neural network model **200** and a desired output from the neural network model for each respective dataset of the input data. The modelling framework **131** can be used to train the neural network model with the training data. An output of the training is the weights and biases that are associated with each node of the neural network model. After the neural network model is trained, the modelling framework **131** can be used to classify new data that is applied to the trained neural network model. Specifically, the trained neural network model uses the weights and biases obtained from training to perform classification and recognition tasks on data that has not been used to train the neural network model. The modelling framework **131** can use the CPU **120** and the special-purpose processors (e.g., the GPU **122** and/or the neural network accelerator **180**) to execute the neural network model with increased performance as compare with using only the CPU **120**. In some examples, the performance can potentially achieve real-time performance for some classification tasks.

The compiler **132** analyzes the source code and data (e.g., the examples used to train the model) provided for a neural network model and transforms the model into a format that can be accelerated on the neural network accelerator **180**, which will be described in further detail below. Specifically, the compiler **132** transforms the source code into executable code, metadata, configuration data, and/or data structures for representing the neural network model and memory as neural network data **200**. In some examples, the compiler **132** can divide the neural network model into portions (e.g., neural network **200**) using the CPU **120** and/or the GPU **122**) and other portions (e.g., a subgraph, an individual layer, or a plurality of layers of a neural network) that can be executed on the neural network accelerator **180**. The compiler **132** can generate executable code (e.g., runtime modules) for executing NNs assigned to the CPU **120** and for communicating with a subgraph, an individual layer, or a plurality of layers of a neural network assigned to the accelerator **180**. The compiler **132** can generate configuration data for the accelerator **180** that is used to configure accelerator resources to evaluate the subgraphs assigned to the optional accelerator **180**. The compiler **132** can create data structures for storing values generated by the neural network model during execution and/or training and for communication between the CPU **120** and the accelerator **180**. The compiler **132** can generate metadata that can be used to identify subgraphs, edge groupings, training data, and various other information about the neural network model during runtime. For example, the metadata can include information for interfacing between the different subgraphs or other portions of the neural network model.

The runtime environment **133** provides an executable environment or an interpreter that can be used to train the neural network model during a training mode and that can be used to evaluate the neural network model in training, inference, or classification modes. During the inference mode, input data can be applied to the neural network model inputs and the input data can be classified in accordance with the training of the neural network model. The input data can be archived data or real-time data.

The runtime environment **133** can include a deployment tool that, during a deployment mode, can be used to deploy or install all or a portion of the neural network to neural network accelerator **180**. The runtime environment **133** can further include a scheduler that manages the execution of the different runtime modules and the communication between the runtime modules and the neural network accelerator **180**. Thus, the runtime environment **133** can be used to control the flow of data between nodes modeled on the neural network module **130** and the neural network accelerator **180**.

In one example, the neural network accelerator **180** receives and returns normal-precision values **150** from the neural network module **130**. As illustrated in FIG. **1**, the quantization accelerator **186** can perform a bulk of its operations using quantized floating-point and an interface between the quantization accelerator **186** and the neural network module **130** can use full-precision values for communicating information between the modules. The normal-precision values can be represented in 16-, 32-, 64-bit, or other suitable floating-point format. For example, a portion of values representing the neural network can be received, including edge weights, activation values, or other suitable parameters for quantization. The normal-precision values **150** are provided to a normal-precision floating-point to quantized floating-point converter **152**, which converts the normal-precision value into quantized values. Quantized floating-point operations **154** can then be performed on the quantized values. The quantized values can then be converted back to a normal-floating-point format using a quantized floating-point to normal-floating-point converter **156** which produces normal-precision floating-point values. As a specific example, the quantization accelerator **186** can be used to accelerate a given layer of a neural network, and the vector-vector, matrix-vector, matrix-matrix, and convolution operations can be performed using quantized floating-point operations and less compute-intensive operations (such as adding a bias value or calculating an activation function) can be performed using normal floating-point precision operations. Other examples of conversions that can be performed are converting quantized and normal-precision floating-point values to normal floating-point or block floating-point values that have lossy or non-uniform mantissa values or outlier mantissa values. For example, conversion to block floating-point format with non-uniform mantissas or outlier mantissa values can be performed to compress activation values, and a conversion from block floating-point format with non-uniform mantissas or outlier mantissa values to normal-precision floating-point or block floating-point can be performed when retrieving compressed activation values for later use, for example, in back propagation.

The conversions between normal floating-point and quantized floating-point performed by the converters **152** and **156** are typically performed on sets of numbers represented as vectors or multi-dimensional matrices. In some examples, additional normal-precision operations **158**, including operations that may be desirable in particular neural network implementations can be performed based on normal-precision formats including adding a bias to one or more nodes of a neural network, applying a hyperbolic tangent function or other such sigmoid function, or rectification functions (e.g., ReLU operations) to normal-precision values that are converted back from the quantized floating-point format.

In some examples, the quantized values are used and stored only in the logic gates and internal memories of the neural network accelerator **180**, and the memory **125** and

storage **129** store only normal floating-point values. For example, the neural network accelerator **180** can quantize the inputs, weights, and activations for a neural network model that are received from the neural network model **130** and can de-quantize the results of the operations that are performed on the neural network accelerator **180** before passing the values back to the neural network model **130**. Values can be passed between the neural network model **130** and the neural network accelerator **180** using the memory **125**, the storage **129**, or an input/output interface (not shown). In other examples, an emulator provides full emulation of the quantization, including only storing one copy of the shared exponent and operating with reduced mantissa widths. Some results may differ over versions where the underlying operations are performed in normal floating-point. For example, certain examples can check for underflow or overflow conditions for a limited, quantized bit width (e.g., three-, four-, or five-bit wide mantissas).

The bulk of the computational cost of DNNs is in vector-vector, matrix-vector, and matrix-matrix multiplications and/or convolutions. These operations are quadratic in input sizes while operations such as bias add and activation functions are linear in input size. Thus, in some examples, quantization is only applied to matrix-vector multiplication operations, which is implemented on the neural network accelerator **180**. In such examples, all other operations are done in a normal-precision format, such as float16. Thus, from the user or programmer's perspective, the quantization-enabled system **110** accepts and outputs normal-precision float16 values from/to the neural network module **130** and output float16 format values. All conversions to and from block floating-point format can be hidden from the programmer or user. In some examples, the programmer or user may specify certain parameters for quantization operations. In other examples, quantization operations can take advantage of block floating-point format to reduce computation complexity, as discussed below regarding FIG. **3**.

The neural network accelerator **180** is used to accelerate evaluation and/or training of a neural network graph or subgraphs, typically with increased speed and reduced latency that is not realized when evaluating the subgraph using only the CPU **120** and/or the GPU **122**. In the illustrated example, the accelerator includes a Tensor Processing Unit (TPU) **182**, reconfigurable logic devices **184** (e.g., contained in one or more FPGAs or a programmable circuit fabric), and/or a quantization accelerator **186**, however any suitable hardware accelerator can be used that models neural networks. The accelerator **180** can include configuration logic which provides a soft CPU. The soft CPU supervises operation of the accelerated graph or subgraph on the accelerator **180** and can manage communications with the neural network module **130**. The soft CPU can also be used to configure logic and to control loading and storing of data from RAM on the accelerator, for example in block RAM within an FPGA.

In some examples, parameters of the neural network accelerator **180** can be programmable. The neural network accelerator **180** can be used to prototype training, inference, or classification of all or a portion of the neural network model **200**. For example, quantization parameters can be selected based on accuracy or performance results obtained by prototyping the network within neural network accelerator **180**. After a desired set of quantization parameters is selected, a quantized model can be programmed into the accelerator **180** for performing further operations.

The compiler **132** and the runtime **133** provide a fast interface between the neural network module **130** and the neural network accelerator **180**. In effect, the user of the neural network model may be unaware that a portion of the model is being accelerated on the provided accelerator. For example, node values are typically propagated in a model by writing tensor values to a data structure including an identifier. The runtime **133** associates subgraph identifiers with the accelerator, and provides logic for translating the message to the accelerator, transparently writing values for weights, biases, and/or tensors to the neural network accelerator **180** without program intervention. Similarly, values that are output by the neural network accelerator **180** may be transparently sent back to the neural network module **130** with a message including an identifier of a receiving node at the server and a payload that includes values such as weights, biases, and/or tensors that are sent back to the overall neural network model.

FIG. **2** illustrates a simplified topology of a deep neural network (DNN) **200** that can be used to perform neural network applications, such as enhanced image processing, using disclosed BFP implementations. One or more processing layers can be implemented using disclosed techniques for quantized and BFP matrix/vector operations, including the use of one or more of a plurality of neural network quantization accelerators **186** in the quantization-enabled system **110** described above. It should be noted that applications of the neural network implementations disclosed herein are not limited to DNNs but can also be used with other types of neural networks, such as convolutional neural networks (CNNs), including implementations having Long Short Term Memory (LSTMs) or gated recurrent units (GRUs), or other suitable artificial neural networks that can be adapted to use BFP methods and apparatus disclosed herein.

The DNN **200** can operate in at least two different modes. Initially, the DNN **200** can be trained in a training mode and then used as a classifier in an inference mode. During the training mode, a set of training data can be applied to inputs of the DNN **200** and various parameters of the DNN **200** can be adjusted so that at the completion of training, the DNN **200** can be used as a classifier. Training includes performing forward propagation of the training input data, calculating a loss (e.g., determining a difference between an output of the DNN and the expected outputs of the DNN), and performing backward propagation through the DNN to adjust parameters (e.g., weights and biases) of the DNN **200**. When an architecture of the DNN **200** is appropriate for classifying the training data, the parameters of the DNN **200** will converge and the training can complete. After training, the DNN **200** can be used in the inference mode. Specifically, training or non-training data can be applied to the inputs of the DNN **200** and forward propagated through the DNN **200** so that the input data can be classified by the DNN **200**.

As shown in FIG. **2**, a first set **210** of nodes (including nodes **215** and **216**) form an input layer. Each node of the set **210** is connected to each node in a first hidden layer formed from a second set **220** of nodes (including nodes **225** and **226**). A second hidden layer is formed from a third set **230** of nodes, including node **235**. An output layer is formed from a fourth set **240** of nodes (including node **245**). In example 200, the nodes of a given layer are fully interconnected to the nodes of its neighboring layer(s). In other words, a layer can include nodes that have common inputs with the other nodes of the layer and/or provide outputs to common destinations of the other nodes of the layer. In other examples, a layer can include nodes that have a subset of common inputs with the other nodes of the layer and/or provide outputs to a subset of common destinations of the other nodes of the layer.

During forward propagation, each of the nodes produces an output by applying a weight to each input generated from the preceding node and collecting the weights to produce an output value. In some examples, each individual node can have an activation function (σ) and/or a bias (b) applied. Generally, an appropriately programmed processor or FPGA can be configured to implement the nodes in the depicted neural network **200**. In some example neural networks, an output function ƒ (n) of a hidden combinational node n can produce an output expressed mathematically as:

$$f(n) = \sigma\left(\sum_{i=0\ to\ E-1} w_i x_i + b\right) \quad \text{(Eq. 2)}$$

where w_i is a weight that is applied (multiplied) to an input edge x_i, b is a bias value for the node n, σ is the activation function of the node n, and E is the number of input edges of the node n. In some examples, the activation function produces a continuous activation value (represented as a floating-point number) between 0 and 1. In some examples, the activation function produces a binary 1 or 0 activation value, depending on whether the summation is above or below a threshold.

A given neural network can include thousands of individual nodes and so performing all of the calculations for the nodes in normal-precision floating-point can be computationally expensive. An implementation for a more computationally expensive solution can include hardware that is larger and consumes more energy than an implementation for a less computationally expensive solution. However, performing the operations using quantized floating-point can potentially reduce the computational complexity of the neural network. A simple implementation that uses only quantized floating-point may significantly reduce the computational complexity, but the implementation may have difficulty converging during training and/or correctly classifying input data because of errors introduced by the quantization. However, quantized floating-point implementations disclosed herein can potentially increase an accuracy of some calculations while also providing the benefits of reduced complexity associated with quantized floating-point.

The DNN **200** can include nodes that perform operations in quantized floating-point. As a specific example, an output function ƒ (n) of a hidden combinational node n can produce an output (an activation value) expressed mathematically as:

$$f(n) = \sigma\left(Q^{-1}\left(\sum_{i=0\ to\ E-1} Q(w_i)Q(x_i)\right) + b\right) \quad \text{(Eq. 3)}$$

where w_i is a weight that is applied (multiplied) to an input edge x_i, $Q(w_i)$ is the quantized floating-point value of the weight, $Q(x_i)$ is the quantized floating-point value of the input sourced from the input edge x_i, $Q^{-1}(\ )$ is the de-quantized representation of the quantized floating-point value of the dot product of the vectors w and x, b is a bias value for the node n, **6** is the activation function of the node n, and E is the number of input edges of the node n. The computational complexity can potentially be reduced (as compared with using only normal-precision floating-point values) by performing the dot product using quantized floating-point values, and the accuracy of the output function can potentially be increased by (as compared with using only quantized floating-point values) by the other operations of the output function using normal-precision floating-point values.

Neural networks can be trained and retrained by adjusting constituent values of the output function $f(n)$. For example, by adjusting weights w_i or bias values b for a node, the behavior of the neural network is adjusted by corresponding changes in the networks output tensor values. For example, a cost function C(w, b) can be used during back propagation to find suitable weights and biases for the network, where the cost function can be described mathematically as:

$$C(w,\ b) = \frac{1}{2n}\sum_x \|y(x) - a\|^2 \tag{Eq. 4}$$

where w and b represent all weights and biases, n is the number of training inputs, a is a vector of output values from the network for an input vector of training inputs x. By adjusting the network weights and biases, the cost function C can be driven to a goal value (e.g., to zero (0)) using various search techniques, for examples, stochastic gradient descent. The neural network is said to converge when the cost function C is driven to the goal value. Similar to the output function $f(n)$, the cost function can be implemented using quantized-precision computer arithmetic. For example, the vector operations can be performed using quantized floating-point values and operations, and the non-vector operations can be performed using normal-precision floating-point values.

Examples of suitable practical applications for such neural network BFP implementations include, but are not limited to: performing image recognition, performing speech recognition, classifying images, translating speech to text and/or to other languages, facial or other biometric recognition, natural language processing, automated language translation, query processing in search engines, automatic content selection, analyzing email and other electronic documents, relationship management, biomedical informatics, identifying candidate biomolecules, providing recommendations, or other classification and artificial intelligence tasks.

A network accelerator (such as the network accelerator **180** in FIG. **1**) can be used to accelerate the computations of the DNN **200**. As one example, the DNN **200** can be partitioned into different subgraphs or network layers that can be individually accelerated. As a specific example, each of the layers **210**, **220**, **230**, and **240** can be a subgraph or layer that is accelerated, with the same or with different accelerators. The computationally expensive calculations of the layer can be performed using quantized floating-point and the less expensive calculations of the layer can be performed using normal-precision floating-point. Values can be passed from one layer to another layer using normal-precision floating-point. By accelerating a group of computations for all nodes within a layer, some of the computations can be reused and the computations performed by the layer can be reduced compared to accelerating individual nodes.

In some examples, a set of parallel multiply-accumulate (MAC) units in each convolutional layer can be used to speed up the computation. Also, parallel multiplier units can be used in the fully-connected and dense-matrix multiplication stages. A parallel set of classifiers can also be used. Such

parallelization methods have the potential to speed up the computation even further at the cost of added control complexity.

As will be readily understood to one of ordinary skill in the art having the benefit of the present disclosure, the practical application of neural network implementations can be used for different aspects of using neural networks, whether alone or in combination or subcombination with one another. For example, disclosed implementations can be used for the practical application of implementing neural network training via gradient descent and/or back propagation operations for a neural network. Further, disclosed implementations can be used for evaluation of neural networks.

IV. Example Quantized Block Floating-Point Formats

FIG. **3** is a diagram **300** illustrating an example of converting a normal floating-point format to a quantized, block floating-point format, as can be used in certain examples of the disclosed technology. For example, input tensors for a neural network represented as normal floating-point numbers (for example, in a 32-bit or 16-bit floating-point format) can be converted to the illustrated block floating-point format.

As shown, a number of normal floating-point format numbers **310** are represented such that each number for example number **315** or number **316** include a sign, an exponent, and a mantissa. For example, for IEEE 754 half precision floating-point format, the sign is represented using one bit, the exponent is represented using five bits, and the mantissa is represented using ten bits. When the floating-point format numbers **310** in the neural network model **200** are converted to a set of quantized precision, block floating-point format numbers, there is one exponent value that is shared by all of the numbers of the illustrated set. Thus, as shown, the set of block floating-point numbers **320** are represented by a single exponent value **330**, while each of the set of numbers includes a sign and a mantissa. However, since the illustrated set of numbers have different exponent values in the floating-point format, each number's respective mantissa may be shifted such that the same or a proximate number is represented in the quantized format (e.g., shifted mantissas **345** and **346**).

Further, as shown in FIG. **3**, use of block floating-point format can reduce computational resources used to perform certain common operations. In the illustrated example, a dot product of two floating-point vectors is illustrated in formal floating-point format (**350**) and in block floating-point format (**360**). For numbers represented in the normal-precision floating-point format operation **350**, a floating-point addition is required to perform the dot product operation. In a dot product of floating-point vectors, the summation is performed in floating-point which can require shifts to align values with different exponents. On the other hand, for the block floating-point dot product operation **360**, the product can be calculated using integer arithmetic to combine mantissa elements as shown. In other words, since the exponent portion can be factored in the block floating-point representation, multiplication and addition of the mantissas can be done entirely with fixed point or integer representations. As a result, large dynamic range for the set of numbers can be maintained with the shared exponent while reducing computational costs by using more integer arithmetic, instead of floating-point arithmetic. In some examples, operations performed by the quantization-enabled system **110** can be optimized to take advantage of block floating-point format.

In some examples, the shared exponent **330** is selected to be the largest exponent from among the original normal-precision numbers in the neural network model **200**. In other examples, the shared exponent may be selected in a different manner, for example, by selecting an exponent that is a mean or median of the normal floating-point exponents, or by selecting an exponent to maximize dynamic range of values stored in the mantissas when their numbers are converted to the quantized number format. It should be noted that some bits of the quantized mantissas may be lost if the shared exponent and the value's original floating-point exponent are not the same. This occurs because the mantissa is shifted to correspond to the new, shared exponent.

There are several possible choices for which values in a block floating-point tensor will share an exponent. The simplest choice is for an entire matrix or vector to share an exponent. However, sharing an exponent over a finer granularity can reduce errors because it increases the likelihood of BFP numbers using a shared exponent that is closer to their original normal floating-point format exponent. Thus, loss of precision due to dropping mantissa bits (when shifting the mantissa to correspond to a shared exponent) can be reduced.

For example, consider multiplying a row-vector x by matrix W: y=xW. If an exponent is shared for each column of W, then each dot-product xW_j (where W_j is the j-th column of W) only involves one shared exponent for x and one shared exponent for W_j.

FIGS. **4** and **5** illustrate alternative block floating-point formats that can be used for computation of neural networks. In the context of neural nets, a core operation is to perform a dot product. For example, dot products are the core computation of matrix multiplication and convolutions. Matrix multiplication involves dot products of the rows/columns of the matrix with an input vector. Convolutions involve dot products of filters with windows of the input. In the context of quantized floating-point, the group of values selected to share an exponent can have an impact on the complexity of the computer arithmetic logic used for calculating the dot product. The values sharing an exponent can be referred to as the values within a bounding box. The shape of bounding boxes can potentially impact quantization error and computation cost. While clustering similar magnitude values to create bounding boxes can reduce quantization error, tracking scaling factors for arbitrary bounding box shapes may be expensive. Instead, matrices and filters can be partitioned into bounding boxes that are potentially more efficient for the operations performed by a neural network. Specifically, an appropriate selection of the bounding box can reduce the complexity of computer arithmetic circuits that are used to implement the operations of the neural network. FIG. **4** illustrates block floating-point formats that may be well suited for matrices and FIG. **5** illustrates block floating-point formats that may be well suited for convolution filters.

FIG. **4** is a diagram **400** illustrating four alternative block floating-point formats, as can be used in certain examples of the disclosed technology. As shown, a first format **410** represents an entire array **420** of values that share a single exponent **425**. In other words, the entire array **420** of values is encapsulated within a single bounding box.

In a second format **430**, a common exponent is shared on a per-column basis. In other words, the columns of the matrix are the bounding box for the values. Thus, in this particular example, block floating-point values stored in even columns **431** of a matrix each share a first, single exponent **432**. Block floating-point values stored in odd columns **435** each share a second, single exponent **437**. In other examples, each column of an array can be associated with a different shared exponent. For an eleven-column tile in the alternative format, there can be eleven corresponding shared exponents, one shared exponent per column. In other examples, each row of an array can be associated with a different shared exponent, or odd and even rows can be associated with a shared common exponent.

A third format **450** is shown where groups of elements in an array share a common exponent. For example, if a 15×15 matrix of values shares in exponent according to the third format **450**, a first set of 5×5 element groups **455** and **456** share a single shared exponent **458**. Similarly, a second 5×5 element group of elements in the array **460** and **461** can each shared a second single exponent **468**. In other examples, each of the tiles can be associated with its own respective shared exponent. In the example format **450**, there could be nine shared exponents for the 15×15 matrix.

A fourth format **470** is shown where two shared exponents are shared on a tiling plus per-column basis. Thus, a first set of numbers including numbers **480**, **481**, and **485** all share a single common exponent **488**. Similarly, a second set of numbers including a set **490** and **491** each share a second, different single exponent **495**. In an alternative example, each of the groups shown can have its own shared exponent.

In some examples, the computational cost of matrix-vector multiplication can be further reduced by reducing mantissa widths. A large range of values having a shared common exponent can be expressed with only a few bits of mantissa. for example, in a representation with four bits of mantissa and a five-bit exponent, values can be expressed in a range $[2^{-14}0.001_2, 2^{15}1.111_2]$, or approximately $[2^{-17}, 2^{16}]$, in contrast, a four-bit fixed point number can only represent values in the range $[0001_2, 1111_2]$, or approximately $[2^0, 2^4]$.

FIG. **5** is a diagram **500** illustrating three alternative block floating-point formats, as can be used in certain examples of the disclosed technology. These formats may be useful for two-dimensional convolutions, but the formats can be generalized to higher-dimensional convolutions as well. As shown, a first format **510** represents an entire convolution filter **512** of values that share a single exponent **514**. A different convolution filter **516** of values can share a single exponent **518**. Thus, the format **510** illustrates that an entire convolution filter can be a bounding box of the values.

In a second format **520**, each spatial pixel can be a bounding box so that a common exponent is shared on a per-spatial-pixel basis, along the channel dimension. As shown, the spatial pixel values **522** share a single exponent **524** and the spatial pixel values **526** share a single exponent **528**. For example, for an input with dimensions $[x, y, c_i]$, each spatial dimension x and y can define a bounding box with c_i values. Similarly, for c_o convolution filters of dimension $[f_x, f_y, c_i]$, each pixel (f_x, f_y) for each of the c_o filters can be a separate bounding box with c_i values. The bounding box size for this approach is c_i.

In a third format **530**, each spatial pixel can be sub-divided along the channel dimension so that a bounding box includes a sub-division of a spatial pixel. As shown, the sub-divided spatial pixel values **532** share a single exponent **534** and the sub-divided spatial pixel values **536** share a single exponent **538**. For small c_i, the cost of handling the scaling factor can be significant. For example, input images at the first layer of deep convolutional neural nets may have $c_i=3$ corresponding to three color channels. Tracking a scaling factor for every triplet can be expensive. In this case, the convolution can be re-shaped into a matrix-matrix multiplication to increase the bounding box and decrease the expense of tracking the bounding box. For example, each convolution filter can be flattened to create a matrix W with c_o columns and f_x, $*f_y$, $*c_i$ rows. An input matrix X can be created where each row is a f_x, $*f_y$, $*c_i$ vector corresponding to a window of the input that the convolution filter sweeps over. The result Y=XW is a matrix that can be re-shaped to match the output of the convolution operation. With the convolution re-formulated as matrix multiplication, the bounding box strategies discussed above in reference to FIG. **4** for matrix multiplication can be applied.

V. Example Methods of Neural Network Training

FIG. **6** is a flow diagram depicting a method **600** of training a neural network using a quantized model, as can be implemented in certain examples of the disclosed technology. For example, training the neural network can include iterating through a set of training data, where the method **600** is used for updating the parameters of the neural network during a given iteration of training data. As one example, the method **600** can be performed by a quantization-enabled system, such as the quantization-enabled system **110** of FIG. **1**.

At process block **610**, parameters, such as weights and biases, of the neural network can be initialized. As one example, the weights and biases can be initialized to random normal-precision floating-point values. As another example, the weights and biases can be initialized to normal-precision floating-point values that were calculated from an earlier training set. The initial parameters can be stored in a memory or storage of the quantization-enabled system. In one example, the parameters can be stored as quantized floating-point values which can reduce an amount of storage used for storing the initial parameters.

At process block **620**, input values of the neural network can be forward propagated through the neural network. Input values of a given layer of the neural network can be an output of another layer of the neural network. The values can be passed between the layers from an output of one layer to an input of the next layer using normal-precision floating-point. The output function of the layer i can include an activation value term that is described mathematically as:

$$y_i = Q^{-1}(f(Q(y_{i-1}), Q(W_i))) \qquad \text{(Eq. 5)}$$

where y_{i-1} is the output from a layer providing the input to layer i, W_i is the weight tensor for the layer i, $f(\ )$ is a forward function of the layer, Q( ) is a quantization function, and $Q^{-1}(\ )$ is a de-quantization function. The output function of the layer can be the de-quantized representation of $f(\ )$, or alternatively, the output function can include additional terms, such as an activation function or the addition of a bias, that are performed using normal-precision floating-point (after de-quantization) or using quantized floating-point (before de-quantization). Generally, the inputs, outputs, and parameters of the layers are tensors. Typically, the inputs, outputs, and parameters of the layers will be vectors or matrices. The quantization function Q( ) converts normal-precision floating-point values to quantized floating-point values. The quantization function can be selected to account for the type of input data and the types of operations performed by the layer i. For example, when y_i and W_i are two-dimensional matrices and the output function includes a term that takes the cross product of y_{i-1} and W_i, the quantization function for y_{i-1} can use a bounding box including a row or a portion of a row of y_{i-1}, and the quantization function for W can use a bounding box including a column or a portion of a column of W_i. The computation can be more efficient when selecting the bounding boxes to follow the flow of the operators, thus making a hardware implementation smaller, faster, and more energy efficient. The de-quantization function $Q^{-1}(\ )$ converts quantized floating-point values to normal-precision floating-point values.

Also at process block **620**, a performance metric can be determined for the neural network. In some examples, the performance metric indicates accuracy of the neural network, for example, based on a set of training data. In some examples, the performance metric is based on at least one of the following metrics: a number of true positives, a number of true negatives, a number of false positives, or a number of false negatives generated by the neural network. In some examples, the performance metric is based on entropy of one or more layers of the neural network. In some examples, the performance metric is based on a rate distortion function.

Also at process block **620**, an adjusted parameter can be selected for the neural network based at least in part on the performance metric determined at process block **650**. For example, any of the parameters initially selected at process block **610** can be adjusted. For example, numerical precision of number formats used represent activation values of the neural network can be adjusted. For example, a number of bits used to represent activation value mantissas can be increased based on the performance metric.

At process block **630**, a portion of a neural network, such as a layer that was just forward propagated to the next layer of the neural network can be compressed and stored in memory. For example, activation values calculated as part of forward propagation as discussed above process block **620** can be compressed and stored in the memory. This compression can be expressed mathematically as:

$$y_{ci} = C(Q2(y_i)) \qquad \text{(Eq. 6a)}$$

or

$$y_{ci} = C(Q2(f(Q(y_{i-1}), Q(W_i)))) \qquad \text{(Eq. 6b)}$$

where y_i are the values generated by forward propagation for a layer at process block **620**, C( ) is an optional, additional compression function (which may include multiple compression operations), Q2( ) is a quantization function to further compress values to a second floating-point format (for example, a format having a fewer number of mantissa bits, a fewer number of exponent bits, a different exponent sharing scheme, having lossy or non-uniform mantissas, or outlier mantissa values for some of the values), and y_{ci} are the compressed values to be stored in memory. At least one parameter of the quantization function Q2( ) can be adjusted based on a performance metric. For example, output of a layer can be compared to expected values (for example, using a confusion matrix or individual components thereof) to measure performance of the neural network. In other examples, one or more of: mean square error, perplexity, gradient signal to noise ratio, or entropy of the neural network can be used. Examples of adjusted activation compression parameters that can be adjusted include number of bits of mantissa, number of bits of exponent, exponent sharing scheme, lossy or non-uniform mantissa scheme, use of outlier values for a subset of the activation values, or any other disclosed activation compression parameter that can be adjusted. Typically, precision of the stored compressed activation values is increased as training of the neural network proceeds, although it is possible in some cases for the precision to be reduced from a prior training batch or epoch, for a portion of the training. In some examples, the Q2( ) quantization function translates values from a normal precision format to a smaller quantized format than used in the quantized layer (as in equation 6a). In other examples, the Q2( ) quantization function translates values directly from the first block floating-point format used in the quantized layer to a second block floating-point format (as in equation 6b).

The compressed activation values are expressed in a second block floating-point format that can differ from a first block floating-point format used to perform forward propagation calculations and at least one of the following ways: having a different mantissa format, having a different exponent format, or having a different exponent sharing scheme. For example, if forward propagation was performed using activation values expressed in an eight-bit format, these values can be transformed to a four-bit format by converting the mantissa to a lossy or non-uniform mantissa. As another example, activation value exponents, including shared exponents in BBFP format can be transformed from a seven-bit format to a five-bit format. As another example, for a four-bit number having a sign bit and three mantissa bits, the three-bit mantissa (which can represent 8 values, for example, the set {0, 1, 2, 3, 4, 5, 6, 7} may be converted to a lossy mantissa having discrete sets of mantissa values, for example, any one of the sets: {0, 1, 3, 7}, {0, 1, 7}, or {0, 7}, depending on a selected lossy mantissa scheme. The underlying representation of the lossy mantissa may vary. For the preceding three sets, an example set of respective binary representations is {00, 01, 10, 11}; {00, 10, 11}; and {0, 1}; respectively. In some examples, some of the mantissas can be converted to an outlier mantissa format, where selected values have additional bits of mantissa stored, as will be further described below. In some examples, multi-value (non-binary) representations can be used to for the underlying representation of the lossy mantissa. For example, multi-level EEPROM or flash devices can store a lossy mantissa using multi-level storage cells.

Values can be translated between the two quantized formats used by any suitable technique. For example, a lookup table, logic gates, arithmetic units, or other circuitry can be used to convert values from a normal-precision floating-point or block floating-point format to a lower precision floating-point format. For examples, mantissas can be truncated or rounded to have fewer bits. Mantissas can be converted to a mantissa format having lossy or non-uniform mantissas. Some of the mantissa values can be converted to a mantissa format having additional bits of mantissa information stored as outlier values. In some examples, the floating-point format is a block floating-point format, where an exponent is shared between two or more values. In other examples, a normal floating-point format, but having lossy or non-uniform mantissas and/or outlier mantissa values, is used.

In some examples, the compressed block floating-point format shares a common exponent in a different way than the format used when performing forward propagation. Aspects of the sharing format that can be changed include how an exponent is shared on a per-row, per-column, or per-tile basis. In some examples, additional compression can be applied to the compressed block floating-point format prior to storing in memory. Examples of suitable techniques for further compressing activation values in the compressed quantized format include entropy compression (e.g., Huffman encoding), zero compression, run length compression, compressed sparse row compression, or compressed sparse column compression.

At process block **640**, a loss of the neural network can be calculated. For example, the output y of the neural network can be compared to an expected output ŷ of the neural network. A difference between the output and the expected output can be an input to a cost function that is used to update the parameters of the neural network.

At process block **650**, activation values stored in memory are decompressed for back propagation, and in particular, for calculation of output error terms used in backpropagation for a particular layer. The method can iterate over each layer and decompress activation values for each layer, perform back-propagation for the layer, and then decompress activation values for the preceding layer. This decompression can be expressed mathematically as:

$$y_i = Q2^{-1}(C^{-1}(y_{ci})) \quad \text{(Eq. 7a)}$$

or

$$y_i = C^{-1}(y_{ci}) \quad \text{(Eq. 7b)}$$

where y_{ci} are the compressed activation values retrieved from memory, $C^{-1}(\ )$ is a decompression function (which may include multiple compression operations) that is inverse of the compression function C( ), $Q2^{-1}(\ )$ is a function that translates quantized values from the second block floating-point format to the first block floating-point format, and y_i are the values generated by forward propagation for a layer at process block **620**. For example, after forward propagation is completed for every layer and a neural network as discussed above regarding process blocks **620** and **630**, and losses calculated as discussed above at process block **640**, values are back propagated back through the neural network, typically starting from the output layer of the neural network. Thus, depending on how the compressed quantized format is different than the format used for back propagation, and appropriate transformation of activation value mantissas, exponents, and/or exponent sharing scheme can be performed. Further, if additional compression was applied prior to storing in memory, such as entropy compression, zero compression, run length encoding, compressed sparse row compression, or compressed sparse column compression, these operations can be reversed prior to performing back propagation for a layer at process block **660**.

A number of different techniques can be used to dequantize lower-precision mantissas. For example, the additional bits in the original (first) floating-point format can be padded with ones or zeros in a deterministic fashion (e.g., padding with all zeros, all ones, all of a selected value) or with random values. Such techniques for replacing additional bits in the original floating-point format can be used when dequantizing outlier values, or mantissas that do not have associated outlier values.

In the case of dequantizing lossy or non-uniform mantissas, a number of different techniques can be used. For example, when the set of non-uniform mantissas is {0, 1, 3, 7} (stored, e.g., in two-bit mantissa format), those same values can be restored as the dequantized values in the first normal-precision (e.g., 8-, 16, or 32-bit) or block floating-point (e.g., three, four, five, or six-bit) mantissa. In some examples, the dequantized mantissa can be an approximation. For example, for the set {0, 1, 3, 7}, the dequantized mantissas can be {0, 1, 3, 5}. In some examples, the dequantized value is selected in a deterministic fashion. For example, for the set {0, 1, 3, 7}, the value 3 can be dequantized to 2 the first time the value is encountered, 3 the second time, 4, the third time, then back to 2 on the fourth time the value is encountered. In some examples, a randomly-selected dequantized mantissa can be used. For example, for the set {0, 1, 3, 7}, the non-uniform mantissa value 3 can be translated to a random value selected from the set {2, 3, 5, 6} and the non-uniform mantissa value 7 can be translated to a random value selected from the set {5, 4, 6, 7}. In some examples, the random value is selected according to a uniform distribution, while in other examples, a normal (Gaussian), Poisson, or other probability distribution is used to select the de-quantized mantissa.

At process block **660**, the loss of the neural network can be back-propagated through the neural network. During back propagation, an output error term ∂y and a weight error term ∂W can be calculated. The output error term can be described mathematically as:

$$\partial y_{i-1}=Q^{-1}(g(Q(\partial y_i),Q(W_i))) \quad \text{(Eq. 8)}$$

where ∂y_{i-1} is the output error term from a layer following layer i, W is the weight tensor for the layer i, g( ) is a backward function of the layer, Q( ) is a quantization function, and Q^{-1}( ) is a de-quantization function. The backward function g( ) can be can be the backward function of f( ) for a gradient with respect to y_{i-1} or a portion of the gradient function. The output error term of the layer can be the de-quantized representation of g( ) or the output error term can include additional terms that are performed using normal-precision floating-point (after de-quantization) or using quantized floating-point (before de-quantization).

The weight error term ∂W can be described mathematically as:

$$\partial W_i=Q^{-1}(h(Q(y_i),Q(\partial y_i))) \quad \text{(Eq. 9)}$$

where ∂W, is the weight error term for the layer i, ∂y_i is the output error term for the layer i, y_i is the output for the layer i, h( ) is a backward function of the layer, Q( ) is a quantization function, and Q^{-1}( ) is an inverse quantization function. The backward function h( ) can be can be the backward function of f( ) for a gradient with respect to W_{i-1} or a portion of the weight error equation 9. The weight error term of the layer can be the de-quantized representation of h( ) or the weight error term can include additional terms that are performed using normal-precision floating-point (after de-quantization) or using quantized floating-point (before de-quantization). The weight error term can include additional terms that are performed using normal-precision floating-point.

At process block **670**, the parameters for each layer can be updated. For example, the weights for each layer can be updated by calculating new weights based on the iteration of training. As one example, a weight update function can be described mathematically as:

$$W_i=W_i+\eta\times\partial W_i \quad \text{(Eq. 10)}$$

where ∂W_i is the weight error term for the layer i, η is the learning rate for the layer i for the neural network, W_i is the weight tensor for the layer i. In one example, the weight update function can be performed using normal-precision floating-point.

VI. Example Environment for Performing Activation Compression

FIG. **7** is a block diagram **700** depicting an example of a suitable environment for performing activation compression and associated floating-point operations between a normal-precision floating-point domain, a quantized floating-point domain, and a floating-point domain having further compressed floating-point formats. The floating-point domain may have exponents for all values, or be a block floating-point format having two or more shared exponents. As described above, more computationally expensive operations such as vector-vector, vector-matrix, matrix-matrix, and convolution operations can be performed by the quantized layer **710** in the quantized floating-point domain. Less computationally expensive operations such as scalar add and scalar multiply can be performed outside of the quantized layer **710** in the normal-precision floating-point domain. With regard to neural networks, a neural network can be partitioned into layers (such as quantized layer **710**). For the back propagation portion of training, an error output component **720** can receive the activation values and use additional functions, such as an error function or an objective function, to calculate the output error term ∂y_i. In some examples, the output error terms ∂y_i are calculated in a normal-precision floating-point domain, as shown by the solid lines to the output error component **720**. In other examples, the output error component **720** calculates the error output terms in a block floating-point format.

In some examples, the bulk of the computational work within a layer can be performed in the quantized floating-point domain and less computationally expensive operations of the layer, such as adding a bias value or calculating an activation function, can be performed in the normal-precision floating-point domain. The values that interface between the layers can be passed from one layer to the other layer in the normal-precision floating-point domain. By quantizing the inputs specifically for a given layer, the quantization can be targeted to the operations of that layer so that the operations of the layer are more efficient. Specifically, bounding boxes of the quantized floating-point format can be selected to reduce the complexity of the computer arithmetic circuits to make the computer logic potentially faster and/or more energy efficient.

As one example, the output values y_i, the output error term ∂y_i, the weights W_i, and the weight error terms ∂W_i for a given layer can be stored in the normal-precision floating-point domain. During the forward propagation flow, the output values from an earlier layer can be communicated from the normal-precision floating-point domain to the quantized floating-point domain through the quantizer **722** that converts from normal-precision floating-point to quantized floating-point. The output values from the given layer can be communicated from the quantized floating-point domain to the normal-precision floating-point domain through the de-quantizer **724** that converts from quantized floating-point to normal-precision floating-point. The weights for the given layer can be communicated from the normal-precision floating-point domain to the quantized floating-point domain through the quantizer **742**. The de-quantized values y_i or the quantized values $Q^{-1}(y_i)$ can be sent to a compressor **760**, which compresses the values before they are stored in a bulk memory **770**.

The compressor **760** can be a block floating-point compressor, or provide other forms of compression to reduce the amount of data stored in the bulk memory **770**. Typically, the first block floating-point format used to represent values during quantized layer **710** operations has uniform mantissas (e.g., in a normal precision floating-point or block floating-point format). For example, the first block floating-point format used in the quantized layer **710** may have mantissas having more bits than the second block floating-point format, for example: four, five, six, seven, or eight bits; and the second block floating-point format may have non-uniform mantissas having fewer bits than the first block floating-point format, for example: three, four, four or five, five or six, or four to six bits, respectively. It should be readily understood to one of ordinary skill in the relevant art having the benefit of the present disclosure that foregoing recited combinations of particular numbers of bits in the first block floating-point format and a second block floating-point format are merely preferred examples, but that other combinations of precision of mantissa format may be used in other examples. In some examples, the second block floating-point format has a lower precision exponent than the first block floating-point format. In some examples, the first block floating-point format uses a sharing format that is different than the sharing format for a common exponent of the second block floating-point format. For example, the sharing format can be different based on per-row, per-column, or per-tile sharing of a common exponent for the compressed activation values. The precision of the second block floating-point format can be adjusted in a number of different ways, including rounding, truncation, and/or shifting of mantissa values.

In the illustrated example, the activation values are dequantized **724** to a normal precision format prior to converting to the second block-floating-point format and storing in the bulk memory **770**. In some examples, the compressor **760** is configured to provide additional compression by further compressing activation values in the second block floating-point format by performing at least one or more of the following compression operations: entropy compression, zero compression, run length encoding, compressed sparse row compression, or compressed sparse column compression.

The bulk memory **770** can be implemented using any suitable memory or storage technology. In some examples, memory storing temporary values in the quantization layer **710** is typically implemented as static ram (SRAM), embedded dynamic RAM (eDRAM), in a register file, in a block RAM, or other suitable structure, while the bulk memory **770** is typically implemented in memory structures supporting larger, but often slower access, for example off-chip DRAM, network accessible RAM, SSD drives, hard drives, or network-accessible storage. As will be readily understood to one of ordinary skill in the relevant art having the benefit of the present disclosure, the types and arrangement of memory used to implement memory for the quantization layer **710** and the bulk memory **770** can be selected depending on desired performance attributes such as costs, energy, and speed.

A decompressor **780** reads the compressed activation values from the bulk memory **770** and reverses the operations performed by the compressor **760**. In those examples where additional compression is applied to the quantized values, the values are decompressed prior to dequantizing. The values can then be translated from the second block floating-point format to a normal precision floating-point format (dequantized). The second block floating-point format values can be dequantized in a number of different ways, such as those discussed above at process block **650** for the method of FIG. **6**. For example, dequantized non-uniform mantissa values can be converted to the same values in a uniform mantissa format. Alternatively, the dequantized mantissa values can be converted to the approximate values in a uniform mantissa format, alternate values chosen in a deterministic fashion, or random values according to a distribution selected for a particular non-uniform mantissa value. In alternative examples, the values are output by the decompressor **780** in a quantized block floating-point format, as indicated by the dashed line.

The output error component **720** can receive activation values in normal precision floating-point format (as in equation 6a). In alternative examples, the output error component **720** can receive activation values in a quantized floating-point format (as in equation 6b), such as the second block floating-point format, as indicated by the dashed line.

The output error component **720** calculates the derivative of the activation value which is used in back propagation. The back propagation is performed using the quantized layer **710** as indicated in the diagram **700**. The output error component **720** can be used to generate a performance metric **725** that is used to select the second block floating-point format used by the compressor. Examples of measures that can be used to generate the performance metric **725** include metrics generated by comparing output of the neural network to expected output values. Such metrics can include one or more of the following: a true positive rate, a true negative rate, a positive predictive rate, a negative predictive value, a false negative rate, a false positive rate, a false discovery rate, a false omission rate, or an accuracy rate.

As will be readily understood to one of ordinary skill in the relevant art having the benefit of the present disclosure, other metrics can be used to supplement, or in addition to, the expected output value metrics discussed in the preceding paragraph. For example, one or more of: mean square error, perplexity, gradient signal to noise ratio, or entropy of the neural network can be used.

As training of a neural network progresses, the accuracy of the second block floating-point format can be adjusted (and typically, increased). Accuracy can be adjusted in a number of different ways, for example by increasing the number of bits used to express mantissa values, by adjusting the number of compressed values that have associated outlier values, by adjusting the manner and number of bits used to express exponents, or by adjusting other parameters of the compressed activation values stored in the bulk memory **740**.

The dashed arrows in the diagram **700** depict an alternative path for compressing and decompressing activation values. As shown, as successive layers of a neural network are forward propagated, producing first activation values in a first block floating-point format, the compressor can convert the quantized activation values from the first block floating-point format directly to a second block floating-point format, thereby producing compressed activation values, without converting to normal precision format.

During the back-propagation flow **730**, the output error terms from a later layer can be communicated from the normal-precision floating-point domain to the quantized floating-point domain through the quantizer **732**. The output error term from the given layer can be communicated from the quantized floating-point domain to the normal-precision floating-point domain through the de-quantizer **734**. The weights for the given layer can be communicated from the normal-precision floating-point domain to the quantized floating-point domain through the quantizer **742**. The weight error term from the given layer can be communicated from the quantized floating-point domain to the normal-precision floating-point domain through the de-quantizer **754**. As back propagation proceeds, quantized activation values in the second block floating-point format are either converted to a normal precision format and translated to the first block floating-point format through the quantizer **732**, or alternatively, are converted from the second block floating-point format used to store the activation values in the bulk memory **770** to the first block floating-point format used by the quantized layer **710**. Further, in examples where additional compression (e.g., entropy coding, zero encoding, or other additional compression scheme) is applied prior to storing in the bulk memory **770**, the data can be further decompressed before the quantizing.

FIG. **8** is a diagram **800** further detailing operation of the compressor **760** discussed above regarding FIG. **7**. As shown in the diagram **800**, activation values y_i are received from a dequantizer **724**. The dequantizer **724** can receive the activation values from local memory used to implement the quantized layer **710**.

As shown in FIG. **8**, set of normal precision floating-point values **810** are received by the compressor **760**, for example from the de-quantizer **724**. These set of normal precision floating-point values **810** are provided to a quantizer **820** which implements a quantization function Q2( ) that converts the normal precision values to a second block floating-point format different than the block floating-point format used by the quantized layer **710**. The second block floating-point format may include non-uniform or lossy mantissas or mantissas having outlier values. In the illustrated example, the normal precision floating-point values **810** each have a 10-bit mantissa, a five-bit exponent (for each value), and a one-bit sign bit. These normal precision floating-point values **810** are converted to a set of values $Q2(y_i)$ in the second block floating-point format **830**. In the illustrated example, values expressed in the second block floating-point format **830** each have a non-uniform three-bit mantissa, a one-bit sign bit, and all share a four-bit exponent. In some examples, the exponent is not shared by all of a set of values, but can be shared on a per-row, per-column, or per-tile basis. As will be readily understood to a person of ordinary skill in the relevant art having the benefit of the present disclosure, any suitable floating-point format, including formats using lossy or non-uniform mantissas, can be used for the second block floating-point format **830**. For example, for a four-bit number having a sign bit and three mantissa bits, the three-bit mantissa (which can represent eight values, for example, the set {0, 1, 2, 3, 4, 5, 6, 7} may be converted to a lossy mantissa having discrete sets of mantissa values, for example, any one of the sets: {0, 1, 3, 7}, {0, 1, 7}, or {0, 7}, depending on a selected lossy mantissa scheme. The underlying representation of the lossy mantissa may vary. For the preceding three sets, an example set of respective binary representations is {00, 01, 10, 11}; {00, 10, 11}; and {0, 1}; respectively. In some examples, multi-value (non-binary) representations can be used for the underlying representation of the lossy mantissa. For example, multi-level EEPROM or flash devices can store a lossy mantissa using multi-level storage cells.

The set of values in the second block floating-point format **830** can in some examples be provided to an additional compression unit **840** to be further compressed prior to storing in the bulk memory **770**. Examples of suitable techniques for further compressing activation values in the compressed quantized format include entropy compression, zero compression, run length compression, compressed sparse row compression, or compressed sparse column compression.

Whether or not the quantized values in the second block floating-point format **830** are subject to additional compression, the compressed activation values y_{ci} are stored in the bulk memory **770**. The bulk memory **770** is typically implemented in memory structures supporting larger, but often slower access, for example off-chip DRAM, network accessible RAM, SSD drives, hard drives, or network-accessible storage. As will be readily understood to one of ordinary skill in the art having the benefit of the present disclosure, the types and arrangement of memory used to implement memory the bulk memory **770** can be selected depending on desired performance attributes such as costs, energy, and speed.

FIG. **8** further illustrates an alternative example apparatus **860** for storing values in a second block floating-point format, where quantized values are received directly from the quantized layer **710**, instead of receiving values in normal-precision floating-point format. In this alternative example apparatus **860**, quantized values $f(Q(y_{i-1}))$ are received directly from the quantized layer in the first block floating-point format **870**, where each of the set of values has a seven-bit mantissa, a one-bit sign bit, and a shared six-bit exponent. In some examples, the shared exponent is not shared by all values in the said, but can be shared with only certain values on a per-row, per-column, or per-tile basis. These values in the first block floating-point format can be converted directly to the second block floating-point format $Q2(f(Q(y_{i-1})))$, as shown. The set of values in the second block floating-point format **830** can also be provided to the additional compression unit **840** prior to being stored in the bulk memory **770**.

As will be readily understood to one of ordinary skill in the relevant art having the benefit of the present disclosure, the illustrated normal-precision and block floating-point formats are not limited to the particular selections of mantissa, exponent, and sign bit with illustrated, and schemes for sharing exponents between values may also be varied, depending on desired aspects of a particular implementation of the compressor **760**. Further, value stored in the bulk memory **770** can be decompressed by reversing the operations shown in the diagram **800**.

A number of different techniques can be used to dequantize lower-precision mantissas. For example, the additional bits in the original (first) floating-point format can be padded with ones or zeros in a deterministic fashion (e.g., padding with all zeros, all ones, all of a selected value) or with random values. Such techniques for replacing additional bits in the original floating-point format can be used when dequantizing outlier values, or mantissas that did not have associated outlier values.

A different number of techniques can be used to dequantize lossy or non-uniform mantissas. For example, when the set of non-uniform mantissas is {0, 1, 3, 7} (stored, e.g., in two-bit mantissa format), those same values can be restored as the dequantized values in the first normal-precision (e.g., 8-, 16, or 32-bit) or block floating-point (e.g., three-, four-, five, or six-bit) mantissa. In some examples, the dequantized mantissa can be an approximation. For example, for the set {0, 1, 3, 7}, the dequantized mantissas can be {0, 1, 3, 5}. In some examples, the dequantized value is selected in a deterministic fashion. For example, for the set {0, 1, 3, 7}, the value 3 can be dequantized to 2 the first time the value is encountered, 3 the second time, 4, the third time, then back to 2 on the fourth time the value is encountered. In some examples, a randomly-selected dequantized mantissa can be used. For example, for the set {0, 1, 3, 7}, the non-uniform mantissa value 3 can be translated to a random value selected from the set {2, 3, 5, 6} and the non-uniform mantissa value 7 can be translated to a random value selected from the set {5, 4, 6, 7}. In some examples, the random value is selected according to a uniform distribution, while in other examples, a normal (Gaussian), Poisson, or other probability distribution is used to select the de-quantized mantissa.

VII. Example Methods of Determining and Using Performance Metrics to Adjust Activation Compression

As training of a neural network progresses, a performance metric can be used to determine when to adjust a compression format used to store activation values. For example, as training progresses and the accuracy of neural network improves, a second floating-point format used to store activation values before back propagation can be adjusted to have greater precision, thereby helping the neural network converge as training progresses. An example of a performance metric that can be used to determine when to adjust parameters of a neural network is entropy. An example of using entropy to calculate a performance metric is illustrated with reference to the chart **900** of FIG. **9**. In the chart **900**, the x-axis is a ratio of a given distortion D to the square of the variance 6^2 and the y-axis is a rate distortion function R(D). The rate distortion function indicates a minimum number of bits that should be used to represent data for the associated layer of the neural network. An example of a suitable rate distortion function R(D) for a memoryless Gaussian source with squared-error distortion is provided by Equation 11:

$$\begin{aligned} R(D) &= \frac{1}{2}\log_2\left(\frac{\sigma_x^2}{D}\right) \quad \text{if } 0 \le D \le \sigma_x^2 \\ R(D) &= 0 \qquad\qquad\qquad \text{if } D > \sigma_x^2 \end{aligned} \tag{Eq. 11}$$

A first solid line **910** in the chart indicates the frontier where reducing the number of bits will cause information loss according to the Shannon theorem, for data having a particular level of variance. As training progresses, and variance of values in a neural network increase, the frontier will shift, as indicated by a second dotted line **920** in the chart. Thus, as shown, as variance and hence entropy of the data increases, more bits should be used to represent values of the neural network with sufficient precision.

As will be readily understood to one of ordinary skill in the relevant art having the benefit of the present disclosure, other suitable rate distortion functions, or other performance metrics can be used. For example, functions based on entropy or change in entropy for one or more or all layers of a neural network can be used as the performance metric. In other examples, metrics based on accuracy can be used as performance metric.

Metrics based on accuracy use known results for a set of training data and compares the known results to values achieved when using a neural network undergoing training. Values in a confusion matrix, also termed an error matrix are used to form various accuracy functions. The values generated by the neural network are compared to known results to generate the following conditions: a true positive, where the output of the neural network being trained corresponds to a positive result in the known output, a true negative, where the output of the neural network being trained matches a negative result in the known output, a false positive, where the output of the neural network being trained indicates a positive result, and the corresponding known output indicates a negative result, and a false-negative, where the output of the neural network being trained indicates a negative result, and the corresponding known output indicates a positive result. As a simple example, if the known outputs for image matching of cats or dogs, when the neural network being trained indicates the image is a cat but the known output indicates that the image is a dog, this is a false positive. If the neural network being trained indicates the image is not a cat, but the node output indicates that the image is indeed a cat, this is a false negative. Similarly, where both the neural network being trained and the known output indicate that the image is a cat, or the image is not a cat, are considered true positives and true negatives, respectively.

Examples of accuracy metrics that can be generated from these for confusion matrix values include the following:

true positive rate, which is the ratio of true positives to the total number of real positives in the test data, also referred to as sensitivity, recall, or hit rate;

true negative rate, which is the ratio of true negatives to the total number, which may also be referred to as specificity or selectivity;

positive predictive value, which is the ratio of true positives to the sum of all true positives and false positives, which may also be referred to as precision;

negative predictive value, which is the ratio of true negatives to the sum of all true negatives and false negatives, which may also be referred to as negative precision;

false-negative rate, which is the ratio of false negatives to the number of real positives in the data, may also be referred to as miss rate;

false positive rate, which is the ratio of false positives to the total number of real negative cases in the data, which may also be referred to as fall-out;

false discovery rate, which is the ratio of false positives to the sum of all false positives and true positives;

false omission rate, which is a ratio of false negatives to the sum of all false negatives and all true negatives; and

accuracy rate, which is the ratio of the sum of all true positives and all true negatives to the sum of all positive cases and all negative cases in the data.

Perplexity is another measure that can be used as a performance metric. As used herein, the term "perplexity" refers to a measure that indicates how "surprising" an output of a neural network is relative to other outputs for a particular training batch or training epoch. In other words, the perplexity measure indicates how difficult an associated sample is to classify with the neural network. Perplexity can be described by the following equation, where z_i is the input sample, $p(z_i)$ is a one-hot vector generated based on a data label, $q(z_i)$ is a prediction of the DNN model, and C is the total number of classes in a given application:

$$px(z_i)=2^{-\Sigma_{j=1}^{C} p_j(z_i) log(q_j(z_i))} \tag{Eq. 12}$$

In some examples, a log of the perplexity value can be used for simplification. At a given training epoch, a low log-perplexity value implies that the sample is a typical sample and that the neural network model is not "surprised" with the particular sample. In other words, the sample has a relatively low loss value. A high perplexity value indicates that the input sample is hard to classify with the current DNN model. As will be readily understood to one of ordinary skill in the relevant art having the benefit of the present disclosure, other suitable measures of perplexity besides the one expressed in Equation 12 may be used.

As will be readily understood to one of ordinary skill in the relevant art having the benefit of the present disclosure, other metrics can be used to determine the accuracy and thus used to determine a performance metric for determining when to adjust parameters of the neural network including neural network topology and floating-point precision. Examples of suitable measures include but are not limited to: mean square error, gradient signal to noise ratio, or entropy.

Accuracy rates can be compared to predetermined values to determine whether to adjust parameters of the neural network. For example, based on which particular epochs the training processes and, whether the accuracy is improving, or the rate at which the accuracy is improving, a performance metric can indicate whether to adjust parameters of the neural network.

VIII. Example Non-Uniform, Lossy Mantissa Formats for Activation Compression

FIGS. **10-13** illustrate examples of non-uniform, lossy mantissa formats that can be used to store compressed activation values in certain examples of the disclosed technology. As will be readily understood to a person of ordinary skill in the relevant art having the benefit of the present disclosure, these examples are provided for illustration purposes, but other lossy non-uniform formats can be used in other examples.

FIG. **10** is a diagram **1000** illustrating an example of converting a uniform, three-bit mantissa format to a non-uniform, four-value lossy mantissa format, which can be dubbed a "lite lossy format." As shown in the illustrated example, a set of uniform values **1010** in the uniform, three-bit mantissa format is mapped to a set of values **1020** in the non-uniform, four-value lossy mantissa format in a non-uniform fashion. When the uniform mantissa value is 0, it converted to a lossy mantissa value 0. When the uniform mantissa value is 1 or 2, the value is converted to a lossy mantissa value of 1. When the uniform mantissa value is any of 3, 4, 5, or 6, the value is converted to a lossy mantissa value of 3. When the uniform mantissa value is 7, the value is converted to a lossy mantissa value of 7. The underlying representation may be a four-bit representation, in other words, 0, 1, 3, and 7 may be represented as binary values 00, 01, 10, or 11, respectively. Other underlying representations may also be used.

FIG. **11** is a diagram **1100** illustrating an example of converting a uniform, three-bit mantissa format to a non-uniform, three-value lossy mantissa format, which can be dubbed a "normal lossy format." As shown in the illustrated example, a set of uniform values **1110** in the uniform, three-bit mantissa format is mapped to a set of values **1120** in the non-uniform, three-value lossy mantissa format in a non-uniform fashion. When the uniform mantissa value is 0, it converted to a lossy mantissa value 0. When the uniform mantissa value is any of 1, 2, 3, 4, 5, or 6, the value is converted to a lossy mantissa value of 1. When the uniform mantissa value is 7, the value is converted to a lossy mantissa value of 7. The underlying representation may be a two-bit representation, in other words, 0, 1, and 7 may be represented as binary values 00, 01, 10, or 11, respectively. In other examples, other representations with varying numbers of bits can be used, for example, 0, 10, and 11 for 0, 1, and 7, respectively. Other underlying representations may also be used.

FIG. **12** is a diagram **1200** illustrating an example of converting a uniform, three-bit mantissa format to a non-uniform, three-value lossy mantissa format, which can be dubbed an "aggressive lossy format." As shown in the illustrated example, a set of uniform values **1210** in the uniform, three-bit mantissa format is mapped to a set of values **1220** in the non-uniform, two-value lossy mantissa format in a non-uniform fashion. When the uniform mantissa value is 0, it converted to a lossy mantissa value 0. When the uniform mantissa value is any one of 1, 2, 3, 4, 5, 6, or 7, the value is converted to a lossy mantissa value of 7. The underlying representation may be a one-bit representation. Other underlying representations may also be used.

FIG. **13** is a diagram **1300** illustrating an example of converting a uniform value expressed as a sign/mantissa value (as a sign bit and a three-bit mantissa) to a non-uniform value, which can be dubbed a "full range lossy format." As shown in the illustrated examples, a set of uniform values **1310** in the uniform format is mapped to a set of values **1320** in a non-uniform, five-value lossy mantissa format in a non-uniform fashion. When the sign/mantissa value is −7, it is converted to a lossy mantissa value of −7. When the sign/mantissa value is any one of −6, −5, −4, −3, or −2, it is converted to a lossy mantissa value of −3. When the sign/mantissa value is any one of −1, 0, or 1, it is converted to a lossy mantissa value of 0. When the sign/mantissa value is any one of two, three, four, five, or six, it is converted to a lossy mantissa value of 3. When the sign/mantissa value is 7, it is converted to a lossy mantissa value of 7. The underlying representation for the lossy sign/mantissa values may be, for example, a three-bit representation. Other underlying representation formats may also be used.

IX. Example Outlier Formats for Activation Compression

FIG. **14** is a block diagram **1400** depicting an example of a suitable environment for performing activation compression and associated floating-point operations between a normal-precision floating-point domain, quantized floating-point domain, and compressed, quantized floating-point domain including outlier values. Similar to the environment described above regarding FIG. **7**, more computationally expensive operations such as vector-vector, vector-matrix, matrix-matrix, and convolution operations can be performed by the quantized layer **1410** in the quantized floating-point domain. Less computationally expensive operations such as scalar add and scalar multiply can be performed outside of the quantized layer **1410** in the normal-precision floating-point domain. With regard to neural networks, a neural network can be partitioned into layers (such as quantized layer **1410**). For the back propagation portion of training, an error output component **1420** can receive the activation values and use additional functions, such as an error function or an objective function, to calculate the output error term ∂y_i. In some examples, the output error terms ∂y_i are calculated in a normal-precision floating-point domain, as shown by the solid lines to the output error component **1420**. In other examples, the output error component **1420** calculates the error output terms in a block floating-point format.

In some examples, the bulk of the computational work within a layer can be performed in the quantized floating-point domain and less computationally expensive operations of the layer, such as adding a bias value or calculating an activation function, can be performed in the normal-precision floating-point domain. The values that interface between the layers can be passed from one layer to the other layer in the normal-precision floating-point domain. By quantizing the inputs specifically for a given layer, the quantization can be targeted to the operations of that layer so that the operations of the layer are more efficient. Specifically, bounding boxes of the quantized floating-point format can be selected to reduce the complexity of the computer arithmetic circuits to make the computer logic potentially faster and/or more energy efficient.

As one example, the output values y_i, the output error term ∂y_i, the weights W_i, and the weight error terms ∂W, for a given layer can be stored in the normal-precision floating-point domain. During the forward propagation flow, the output values from an earlier layer can be communicated from the normal-precision floating-point domain to the quantized floating-point domain through the quantizer **1422** that converts from normal-precision floating-point to quantized floating-point. The output values from the given layer can be communicated from the quantized floating-point domain to the normal-precision floating-point domain through the de-quantizer **1424** that converts from quantized floating-point to normal-precision floating-point. The weights for the given layer can be communicated from the normal-precision floating-point domain to the quantized floating-point domain through the quantizer **1442**. The dequantized values y_i or the quantized values $Q^{-1}(y_i)$ can be sent to a compressor **1460**, which compresses the values before they are stored in a bulk memory **1470**.

The compressor **1460** can be a block floating-point compressor, or provide other forms of compression to reduce the amount of data stored in the bulk memory **1470**. In such examples, the second block floating-point format has a lower precision mantissa than the first block floating-point format used to represent values during quantized layer **1410** operations. For example, the first block floating-point format used in the quantized layer **1410** may have mantissas having more bits than the second block floating-point format, for example: four, five, six, seven, or eight bits; and the second block floating-point format may have mantissas having fewer bits than the first block floating-point format, for example: three, four, four or five, five or six, or four to six bits, respectively. In the example discussed below, the first block floating point format has six-bit mantissas and the second block floating-point format has four-bit mantissas. It should be readily understood to one of ordinary skill in the relevant art having the benefit of the present disclosure that foregoing recited combinations of particular numbers of bits in the first block floating-point format and a second block floating-point format are merely preferred examples, but that other combinations of precision of mantissa format may be used in other examples.

The compressor **1460** includes an outlier quantizer **1465** that identifies some, but not all of, the activation values as being outliers. The outlier quantizer **1465** can include an exponent selector that determines a shared exponent for the compressed activation values. For example, the selector can identify a shared exponent by determining that at least one of a mean (average), a median, and/or a mode for at least a portion of the activation values. In some examples, the selector can identify a shared exponent by identifying a group of largest outlying values in the set of activation values and for the remaining group of activation values not identified to be in the group of the largest outliers, determining a shared exponent by selecting the largest exponent of the remaining group. In some examples, the exponent used by the largest number of activation values is selected as the shared exponent. The outlier quantizer **1465** also identifies a limited number of the activation values as having outlier values. Those activation values having outlier values will have additional data stored to increase precision and or allow for larger or more precise values to be stored for those particular activation values. For example, a predetermined number of the largest values in the set of activation values can be determined to be outliers. As a non-limiting example, the largest 8 out of 256 values can be selected as outlier activation values. The selected largest eight values will have a second outlier value mantissa and an associated exponent that can be combined when decompressing the compressed outlier activation value.

The number of outlier values can be selected based on performance or hardware attributes. Generally, storing fewer activation values with outlier values decreases storage costs, while storing more activation values with outlier values increases precision of the neural network. In some examples, a predetermined number (e.g., one or a few) of activation values having a shared exponent are stored with outlier values. In some examples, the number of activation values stored with outlier values is determined per memory access. For example, if 128 activation values are stored per memory access, then a predetermined number (e.g., one or a few) of

activation values having outlier values are stored per each memory access. In some examples, the number of activation values stored with outlier values is predetermined based on computation granularity. For example, if 16 activation values are generated for a given unit of computation (e.g., a single clock cycle, or a predetermined number of clock cycles), then a predetermined number (e.g., one or a few) of activation values having outlier values are stored per compute cycle.

In some examples, the second block floating-point format has a same or greater precision exponent than the first block floating-point format. In some examples, the second block floating-point format has a lower precision exponent than the first block floating-point format. In some examples, the first block floating-point format uses a sharing format that is different than the sharing format for a common exponent of the second block floating-point format. For example, the sharing format can be different based on per-row, per-column, or per-tile sharing of a common exponent for the compressed activation values. The precision of the second block floating-point format can be adjusted in a number of different ways, including rounding, truncation, and/or shifting of mantissa values.

In the illustrated example, the activation values can alternatively be dequantized **1424** to a normal precision format prior to converting to the second block-floating-point format and storing in the bulk memory **1470**. In some examples, the compressor **1460** is configured to further compress activation values in the second block floating-point format by performing at least one or more of the following compression operations: entropy compression, zero compression, run length encoding, compressed sparse row compression, or compressed sparse column compression. In some examples, block floating-point data for the activation values y_{ci} is stored in the same general area of the bulk memory **1470** as the outlier data o_{ci}. In other examples, different memory units and/or different types of memory units are used to store the activation value data y_{ci} and the outlier value data o_{ci}. The outlier data o_{ci} can further include index information indicating which of the activation values the outlier values are associated with, and in some examples an individual exponent for each of the outlier values. In other examples, the outlier values o_{ci} can share an exponent, similar to other block floating-point formats.

The bulk memory **1470** can be implemented using any suitable memory or storage technology. In some examples, memory storing temporary values in the quantization layer **1410** is typically implemented as static ram (SRAM), embedded dynamic RAM (eDRAM), in a register file, in a block RAM, or other suitable structure, while the bulk memory **1470** is typically implemented in memory structures supporting larger, but often slower access, for example off-chip DRAM, network accessible RAM, SSD drives, hard drives, or network-accessible storage. As will be readily understood to one of ordinary skill in the art having the benefit of the present disclosure, the types and arrangement of memory used to implement memory for the quantization layer **1410** and the bulk memory **1470** can be selected depending on desired performance attributes such as costs, energy, and speed.

A decompressor **1480** reads the compressed activation values from the bulk memory **1470** and reverses the operations performed by the compressor **1460**. In examples where additional compression is applied to the quantized values, the values are decompressed. The values can then be translated from the second block floating-point format to a normal precision floating-point format. The decompressor

1480 includes an outlier dequantizer **1485** that transforms values from the second block floating-point format back to the first block floating-point format or a normal precision floating-point format. For those activation values having an associated outlier value, the outlier dequantizer **1485** identifies such values using an outlier index and combines the stored outlier value o_{ci} with the block floating-point data y_{ci} (e.g., by adding the two values) to restore the respective activation value. The outlier dequantizer **1485** can pad one or more bits of the decompressed mantissas, with all one or all zero values. In other examples, the outlier dequantizer **1485** selects random values or chooses a median value (e.g., the value three (0011) or the value four (0100) to replace four bits of dropped mantissa) to replace the missing mantissa bits for values that did not have an associated outlier value.

In alternative arrangements, the values are output by the decompressor **1480** in a quantized block floating-point format, as indicated by the dashed line.

The output error component **1420** can receive activation values in normal precision floating-point format (as in equation 6a). In alternative examples, the output error component **1420** can receive activation values in a quantized floating-point format (as in equation 6b), such as the second block floating-point format, as indicated by the dashed line. The output error component **1420** calculates the derivative of the activation value which is used in back propagation. The back propagation is performed using the quantized layer **1410** as indicated in the diagram **1400**. The output error component **1420** can be used to generate a performance metric **1425** that is used to select the second block floating-point format used by the compressor. Examples of measures that can be used to generate the performance metric **1425** include metrics generated by comparing output of the neural network to expected output values. Such metrics can include one or more of the following: a true positive rate, a true negative rate, a positive predictive rate, a negative predictive value, a false negative rate, a false positive rate, a false discovery rate, a false omission rate, or an accuracy rate.

As will be readily understood to one of ordinary skill in the relevant art having the benefit of the present disclosure, other metrics can be used to supplement, or in addition to, the expected output value metrics discussed in the preceding paragraph. For example, one or more of: mean square error, perplexity, gradient signal to noise ratio, or entropy of the neural network can be used.

As training of a neural network progresses, an accuracy parameter of the second block floating-point format can be adjusted (and typically, increased). The accuracy parameter can be adjusted in a number of different ways, for example by increasing the number of bits used to express mantissa values, by adjusting the number of compressed values that have associated outlier values, by adjusting the manner and number of bits used to express exponents, or by adjusting other parameters of the compressed activation values stored in the bulk memory **1470**.

The dashed arrows in the diagram **1400** depict an alternative path for compressing and decompressing activation values. As shown, as successive layers of a neural network are forward propagated, producing first activation values in a first block floating-point format, the compressor can convert the quantized activation values to a normal-precision floating-point format prior to converting to a second block floating-point format, thereby producing compressed activation values.

During the back-propagation flow **1430**, the output error terms from a later layer can be communicated from the normal-precision floating-point domain to the quantized floating-point domain through the quantizer **1432**. The output error term from the given layer can be communicated from the quantized floating-point domain to the normal-precision floating-point domain through the de-quantizer **1434**. The weights for the given layer can be communicated from the normal-precision floating-point domain to the quantized floating-point domain through the quantizer **1442**. The weight error term from the given layer can be communicated from the quantized floating-point domain to the normal-precision floating-point domain through the de-quantizer **1454**. As back propagation proceeds, quantized activation values in the second block floating-point format are either are converted from the second block floating-point format used to store the activation values in the bulk memory **1470** to the first block floating-point format used by the quantized layer **1410** or alternatively, converted to a normal precision format and translated to the first block floating-point format through the quantizer **1432**. Further, in examples where additional compression (e.g., entropy coding, zero encoding, or other additional compression scheme) is applied prior to storing in the bulk memory **1470**, the data can be further decompressed before the quantizing.

FIG. **15** is a diagram **1500** illustrating an example of converting activation values to a second block floating-point format having outlier values for selected activation values. For example, a quantization system **110** as discussed above including use of a compression environment shown in the diagram **1400** of FIG. **14** can be used to perform the illustrated operations.

As shown in FIG. **15**, a set of activation values y_i **1510** is received. Each of the set of activation values has an individual exponent and a six-bit mantissa. A shared exponent 2^5, is selected by analyzing the set of activation values. In this particular example, it is determined that using the selected the shared exponent 2^5 **1520** will allow expression of the most number of activation values **1510** without loss of an unacceptable amount of data. Two outlier activation values **1530** and **1531** are also identified that are substantially larger than other values in the set of activation values. These two outlier activation values will have ancillary outlier data stored, allow for larger values than those that can be expressed using the shared exponent **1520**.

Mantissas for all of the activation values **1510** are shifted **1540** based on the shared exponent **1520**. For all of the compressed activation values, as set of truncated, N-bit block floating-point mantissas y_{ci} **1550** (in the illustrated example, four bits) is generated for the portions of the shifted mantissas **1540** associated with the shared exponent **1520**, for both non-outliers and outlier activation values. For the two outlier activation values, another N additional bits of mantissa values **1560** (in the illustrated example, four bits) and one of a number of outlier indices **1570** are generated for each outlier value o_{ci}. The exponent associated with each outlier value can be determined a number of different ways. In some examples, the outlier value mantissas **1560** are simply a predetermined number of bits of mantissa to the left of the bits of mantissa for all of the activation values. In other words, the next N more significant bits of mantissa are selected as the outlier value mantissas. In some examples, the outlier value mantissas are associated with a shared exponent for two or more of the outlier values. In some examples, each of the outlier value mantissas **1560** is associated with an individual exponent. In the illustrated example, the outlier values share the same outlier exponent 2^1 **1575**. Each of the outlier indices **1570** indicates which activation value the respective outlier mantissa is associated with. The illustrated example, the outlier values are associated with the activation values from rows 0 and 3. Thus, by storing an index for each of the outlier values, memory is only used to store additional bits of mantissa for those activation values that have an outlier value.

FIG. **16** is a diagram **1600** further detailing operation of the compressor **1460** discussed above regarding FIG. **14**. As shown in the diagram **1600**, quantized activation values $f(Q(y_{i-1})$ **1610** are received from the quantized layer **1410**. Alternatively, the dequantizer **1424** can produce normal precision floating point values **1620** based on activation values received from local memory used to implement the quantized layer **1410**.

As shown in FIG. **16**, the quantized values are expressed in a first block floating-point format **1610** including a six-bit shared exponent, a sign bit for each individual value, and a six-bit mantissa for each individual value. In some examples, the exponent is not shared by all of a set of values, but can be shared on a per-row, per-column, or per-tile basis.

In the alternative example where a set of normal precision floating-point values **1620** are received by the compressor **1460**, these values are provided to a quantizer **1625** which implements a quantization function Q2( ) that converts the normal precision values to a second block floating-point format, different than the block floating-point format used by the quantized layer **1410**. In the illustrated alternative example, the normal precision floating-point values **1620** each have a 10-bit mantissa, a five-bit exponent (for each value), and a one-bit sign bit. These normal precision floating-point values **1610** are converted to a set of values $Q2(y_i)$ in the first block floating-point format **1630**, each have a six-bit mantissa, a one-bit sign bit, and all sharing a six-bit exponent. In some examples, the exponent is not shared by all of a set of values, but can be shared on a per-row, per-column, or per-tile basis.

The outlier quantizer **1465** receives the quantized values in the first block floating-point format **1610** and identifies a shared exponent that will be shared by mantissas for all of the activation values. As shown, the values from the first block floating-point format **1610** are converted to a second block floating-point format **1620**, which has a shared exponent, and each of the values has a one-bit sign and a three-bit mantissa. The shared exponent can be selected a number of different ways. For example, the most common exponent for the set of activation values can be selected as the shared exponent. In some examples, a mean, median, or mode is used to select the shared exponent. In some examples, the shared exponent is selected based on available storage for outlier values. In other words, selected number of outlier values are identified, and a shared exponent suitable for storing the remaining activation values is selected. In some examples, the exponent is not shared by all of a set of values, but can be shared on a per-row, per-column, or per-tile basis. The second block floating-point format further includes data for a number of outlier values. For each identified outlier value, there is an outlier mantissa. In some examples, there is a shared outlier exponent for all of the outlier values, or a portion of outlier values selected on a per-row, per-column or per-tile basis. In some examples, each of the outlier values has its own individual exponent. Further, the outlier values are associated with an outlier index that can be used to identify which of the activation values the outlier value is associated with. The outlier indices can be used when decompressing the activation outlier values.

The set of values in the second block floating-point format **1630** can in some examples be provided to additional compression units **1660** and **1661**. Examples of suitable techniques for further compressing activation values in the compressed quantized format include entropy compression, zero compression, run length compression, compressed sparse row compression, or compressed sparse column compression.

Whether or not the quantized values in the second block floating-point format **1630** are subject to additional compression, the compressed values y_{ci} and outlier values o_{ci} are stored in the bulk memory **1470**. The bulk memory **1470** is typically implemented in memory structures supporting larger, but often slower access, for example off-chip DRAM, network accessible RAM, SSD drives, hard drives, or network-accessible storage. As will be readily understood to one of ordinary skill in the art having the benefit of the present disclosure, the types and arrangement of memory used to implement memory the bulk memory **1470** can be selected depending on desired performance attributes such as costs, energy, and speed. In some examples, the outlier values are stored in a different memory or a different portion of the bulk memory **1470**.

As will be readily understood to one of ordinary skill in the relevant art having the benefit of the present disclosure, the illustrated normal-precision and block floating-point formats are not limited to the particular selections of mantissa, exponent, and sign bits illustrated, and schemes for sharing exponents between values may also be varied, depending on desired aspects of a particular implementation of the compressor **1460**. Further, value stored in the bulk memory **1470** can be decompressed by reversing the operations shown in the diagram **1600**.

X. Example Outlier Quantization Apparatus

FIG. **17** is a block diagram **1700** outlining an example outlier quantizer **1710** as can be implemented in certain examples of the disclosed technology. For example, the illustrated outlier quantizer can be used to perform operations associated with the outlier quantizer's discussed above regarding FIGS. **14-16** in, for example, the quantization enabled system **110** discussed above.

As shown in FIG. **17**, the outlier quantizer **1710** receives a number of activation values expressed in a floating-point format. The floating-point format could be normal precision floating-point or a first block floating-point format. An outlier selector **1725** analyzes the activation values to determine a shared exponent **1730**. For each of the activation values **1720**, the selected shared exponent **1730** is compared to each **1745** of the activation values using a comparator **1740**. If a particular activation value **1745** has a different exponent than the shared exponent **1730**, then the shift controller **1750** selects a shift amount and the shifter **1755** is used to shift the mantissa of the activation value left or right, depending on the selected shared exponent **1730**.

In the illustrated example, four bits of each of the activation value y_{ci} mantissas are stored as activation value mantissas **1761** in a block floating-point memory **1760**. The shift controller **1750** can be used to configure the shifter **1755** to align the four bits of activation value mantissa to be stored in the block floating-point memory **1760** by shifting **1757** the mantissa left or right, based on determinations made by the comparator **1740** when comparing the selected shared exponent and the exponent of the particular activation value **1745**.

The outlier selector **1725** also determines which of the activation values **1720** will be selected to have additional bits of mantissa stored as outlier values o_{ci}. The shift controller **1750** configures the shifter **1755** such that the other bits of mantissa can be selected as the outlier mantissa values. These outlier mantissa values **1771** are stored in an outlier memory **1770**. The outlier quantizer **1710** also includes an address register **1775** that is used to select activation values. The current address of the selected, particular activation value **1745** is used to store an indicator **1777**, also called an outlier index, in the outlier memory **1770**. In some examples, the outlier memory **1770** also stores an individual exponent **1779** for each of the outlier values as shown in dashed lines. In other examples, the outlier values are associated with a shared exponent, or are assumed to be more significant bits of the mantissas in the first block floating-point format. In the illustrated example, for example, the three bits to the left of the decimal point can simply be stored in the outlier memory **1770**, without explicitly storing a shared exponent. In such cases, particular outlier value mantissas may need to be additionally shifted according to the exponent of a given activation value.

Thus, the illustrated outlier quantizer **1710** can maintain higher precision for selected outlier values, while still allowing for compressed storage of mantissas for the bulk of the activation values. In some examples, the quantizer selects a fixed number of outlier values for a group of activation values. For example, every four values out of 64 can be selected as outlier values. In other examples, the outlier values and associated indices and/or exponents can be stored using a data structure that allows for an arbitrary number of outlier values, for example a stack or a queue. As will be readily understood to one of ordinary skill in the relevant art having the benefit of the present disclosure, different configurations and numbers of outlier values and compressed activation values can be selected. In some examples, the outlier memory **1770** and the block floating-point memory **1760** are formed from the same storage device or memory. In other examples, the block floating-point memory **1760** may be formed from a different memory structure and/or type of memory than the memory used to form the outlier memory **1770**.

XI. Example Method of Storing Compressed Activation Values

FIG. **18** is a flowchart **1800** outlining an example method of storing compressed activation values, as can be performed in certain examples of the disclosed technology. For example, the quantization-enabled system discussed above regarding FIG. **1** can be used to implement the illustrated method. In some examples, quantization-enabled systems including the example environments of FIGS. **7**, **8**, **14**, and/or **15** can be used. In some examples, the compressed activation values are stored in a normal-precision floating-point format. In some examples, the compressed activation values are in a block floating-point format. In some examples, the compressed activation values include lossy or non-uniform mantissas. In some examples, the compressed activation values include outlier value mantissas.

At process block **1810**, an activation compression format is selected based on a training performance metric for a neural network. In some examples, the training performance metric is produced by calculating accuracy or changes in accuracy of at least one layer of the neural network. In some examples, the training performance metric is evaluated after compressing activation values in a first activation compression format, proceeding with training of the neural network, evaluating the training performance metric for that further trained neural network, and selecting a second, adjusted activation compression format having increased precision. In some examples, the adjusted activation compression format has a greater number of mantissa bits than the first activation compression format. Some examples, the adjusted activation compression format has a different exponent format than the first activation compression format activation compression format. In some examples, the first activation compression format includes mantissas having outlier values, and the adjusted activation compression format does not comprise mantissas having outlier values. Some examples, the first activation compression format includes non-uniform or lossy mantissas, and the adjusted activation compression format does not have non-uniform or lossy mantissas. In some examples, the first activation compression format activation compression format as a block floating-point format and the adjusted activation compression format is a normal precision floating-point format, where fewer, or none of the mantissas share a common exponent. In some examples, the training performance metric is based on comparing accuracy of the trained neural network to expected values for the output of the neural network. In some examples, the training performance metric is based at least in part on one or more of the following: mean square error, perplexity, gradient signal to noise ratio, or entropy of the neural network.

At process block **1820**, activation values are stored in a computer-readable memory or storage device in the adjusted activation compression format. For example, as layers of a neural network are progressively trained, activation values for a particular layer can be transformed from the format used in performing the training to the adjusted activation compression format and stored in memory. In some examples, the activation values are stored in a bulk-type memory. The stored activation values can then be retrieved after forward propagation has proceeded and decompressed to be used in back propagation operations. In some examples, additional compression operations are performed prior to storing the compressed activation values, for example, entropy compression.

XII. Example Method of Storing Compressed Activation Values in a Selected Floating-Point Format

FIG. **19** is a flowchart **1900** outlining an example method of storing compressed activation values, as can be performed in certain examples of the disclosed technology. For example, the quantization-enabled system discussed above regarding FIG. **1** can be used to implement the illustrated method. In some examples, quantization-enabled systems including the example environments of FIGS. **7**, **8**, **14**, and/or **15** can be used. In some examples, the compressed activation values are stored in a normal-precision floating-point format. In some examples, the compressed activation values are in a block floating-point format. In some examples, the compressed activation values include lossy or non-uniform mantissas. In some examples, the compressed activation values include outlier value mantissas.

At process block **1910**, a neural network is produced including activation values expressed in a first floating-point format. For example, the floating-point format can express activation values in normal-precision floating-point or block floating-point formats. In some examples, the neural network is provided by receiving values for a previously-trained neural network. In other examples, the neural network as provided by performing training for an untrained neural network. As training progresses for successive layers of the neural network, activation values are produced that can be provided for compression and stored, where they may be retrieved for use during back propagation operations. In some examples, hardware circuitry is configured to provide a floating-point compressor. The floating-point compressor can be formed from a processor coupled to memory, a special purpose processor coupled to memory, programming reconfigurable logic devices, such as an FPGA, or implementing devices in an Application Specific Integrated Circuit (ASIC).

At process block **1920**, a second floating-point format is selected based on a performance metric for the neural network. For example, accuracy of the neural network can be evaluated by comparing output of the trained neural network to expected output values to determine one or more accuracy metrics. The accuracy metric may indicate that the accuracy of the neural network has plateaued or is realizing diminishing returns with further training of the neural network, and so the precision of the second floating-point format can be increased so as to further fine-tune accuracy of the neural network. Thus, early training epochs can proceed at a faster rate, using fewer computer resources, while the precision of the compressed activation values can be increased based on the accuracy metric as training converges the neural network during later training epochs.

At process block **1930**, activation values are converted to the second floating-point format selected at process block **1920**. For example, mantissa values for the format used to express the activation values during training can be shifted, truncated, or rounded to a lower precision mantissa format prior to storing the compressed activation values for later use during back propagation. As another example, mantissas in the first floating-point format used for the neural network can be converted to a lossy or non-uniform mantissa format. As another example, mantissas and the first floating-point format used for the neural network can be converted such that some of the mantissas have associated outlier values that are stored with the compressed activation values. As another example, values in the first floating-point format can be in a normal-precision floating-point format and the compressed activation values can be in a block floating-point format. As another example, schemes used to share exponents amongst activation values can be adjusted in the second floating-point format.

At process block **1940**, activation values converted to the second floating-point format are stored in computer readable memory or storage devices. These compressed activation values can be later retrieved for use during back propagation, for example. For example, as layers of a neural network are progressively trained, activation values for a particular layer can be transformed from the format used in performing the training to the adjusted activation compression format and stored in memory. In some examples, the activation values are stored in a bulk-type memory. The stored activation values can then be retrieved after forward propagation has proceeded and decompressed to be used in back propagation operations. In some examples, additional compression operations are performed prior to storing the compressed activation values, for example, entropy compression.

XIII. Example Method of Adjusting Activation Compression Parameters

FIG. **20** is a flowchart **2000** outlining an example method of training a neural network including selecting adjusted activation compression parameters by evaluating the neural network as training progresses. For example, the quantization-enabled system discussed above regarding FIG. **1** can be used to implement the illustrated method. In some examples, quantization-enabled systems including the example environments of FIGS. **7**, **8**, **14**, and/or **15** can be used. In some examples, the compressed activation values are stored in a normal-precision floating-point format. In some examples, the compressed activation values are in a block floating-point format. In some examples, the compressed activation values include lossy or non-uniform mantissas. In some examples, the compressed activation values include outlier value mantissas.

At process block **2010**, training of a neural network begins by performing forward propagation and/or back propagation for at least one training epoch. As training progresses, activation values for one or more layers of the neural network will be determined. These activation values can be stored and then will be later used during back propagation. In some examples, a neural network is at least partially trained prior to being received by a quantization-enabled system. In other examples, training begins with an untrained neural network.

A process block **2020**, accuracy and/or performance properties of a neural network or evaluated. For example, accuracy based on a number of true positives, a number of true negatives, a number of false positives, or number of false negatives can be calculated. In other examples, combinations of these values can be evaluated, such as with ratios. Some examples, other metrics such as false discovery rate, false submission rate, or accuracy rate are used to evaluate performance of the neural network. In some examples, a performance property of the neural network, such as a measure based on mean square error, perplexity, gradient signal to noise ratio, or entropy can be used to evaluate performance of the neural network.

At process block **2030**, an adjusted activation compression parameter is selected based on the evaluation performed at process block **2020**. For example, if accuracy or other performance property of the neural network is not improving in comparison to a predetermined sufficient rate, then the accuracy of the selected, adjusted activation compression parameter can be increased. On the other hand, if the accuracy or other performance property of the neural network is still improving with training, the activation compression parameter can be maintained for number of additional training cycles or epochs.

For example, after a number of training batches have been performed, or a number of training epochs have been performed, performance metrics can be evaluated and a determination made as to whether to adjust the compression format used to store the activation values. Typically, it is desirable to allow for large searching of the neural network solution surface at the beginning of training, and reduce the neural network solution surface by adjusting the neural network precision parameters. For example, a lower precision number format such as a two- or three-bit precision block floating-point format can be used early in training and the precision of numbers used to represent activation values can be increased for successive training operations, as indicated by the performance metric.

One or more aspects of the activation compression parameter can be adjusted. For example, in of the operations described below regarding process blocks **2040**-**2043** can be used in adjusting the activation compression parameter.

At process block **2040**, mantissa widths used to store compressed activation values is adjusted. For example, the number of bits of mantissa used to store compressed activation values can be increased, thereby increasing accuracy of the stored activation values. For example, a four-, six, or eight-bit mantissa values in a previous activation compression format can be increased to eight-, 16-, or 32-bit mantissa values. As will be readily understood to one of ordinary skill in the relevant art having the benefit of the present disclosure, the number of bits of mantissa can be selected in any suitable manner.

At process block **2041**, and exponent sharing scheme can be adjusted. For example, activation compression values previously stored in a block floating-point format can be stored in a normal-precision floating-point format, where none of the exponents are shared. In other examples, an increased number of exponents are shared by a fewer number of mantissa values by adjusting shared tiles of mantissas of a block floating-point array.

At process block **2042**, a lossy mantissa format used to store activation values can be adjusted. For example, a lossy or non-uniform mantissa format can be used in earlier training cycles for the neural network, and a less aggressive mantissa format can be used for subsequent training cycles, based on the accuracy or performance property of the neural network. For example, as training progresses, an aggressive lossy format initially used can be adjusted to a normal lossy format used to store activation values in later training cycles, and then a "lite" lossy format can be used to store mantissa values for even later training. The lossy mantissa format can also be adjusted by storing the compressed activation values without a lossy mantissa in subsequent training cycles, after having previously stored mantissa values in a lossy mantissa format.

At process block **2043**, and outlier value format used to store mantissas for compressed activation values can be adjusted. For example, by adjusting the number of outlier values stored for a set of compressed activation values, adjusting the number of bits used to store the outlier values, or adjusting a scheme used to select outlier values, the outlier value format can be adjusted prior to storing the compressed activation values in the adjusted activation compression format.

At process block **2050**, activation values are compressed to a floating-point format having one or more of the adjusted activation compression parameters that were selected at process blocks **2040-2043**. For example, activation values that are expressed in a first floating-point format during training can be converted to the adjusted activation compression format after forward propagation has been performed for all or a portion of a layer of the neural network. In some examples, modification to floating-point formats can be performed as values are transferred as a part of training. For example, activation values being transferred from a first layer of a neural network to a successive layer of the neural network can be transformed as part of the transfer process.

At optional process block **2060**, additional compression is applied to the activation values converted to the adjusted activation compression format prior to storing in memory. Examples of suitable techniques for further compressing activation values in the compressed quantized format include entropy compression (e.g., Huffman encoding), zero compression, run length compression, compressed sparse row compression, or compressed sparse column compression.

At process block **2070**, activation values converted to the adjusted activation compression format are stored in computer readable memory or storage devices. These compressed activation values can be later retrieved for use during back propagation, for example. For example, as layers of a neural network are progressively trained, activation values for a particular layer can be transformed from the format used in performing the training to the adjusted activation compression format and stored in memory. In some examples, the activation values are stored in a bulk-type memory. The bulk memory can be implemented in any suitable storage technology, including, for example, on- or

off-chip DRAM, network accessible RAM, SSD drives, hard drives, or network-accessible storage. In some examples, the bulk memory is situated on a different integrated circuit than a hardware accelerator used process block floating-point values. In some examples, the bulk memory is situated on the same integrated circuit as a hardware accelerator used to process block floating-point values. After activation values are stored in the bulk memory, computations for a neural networks can proceed to a next layer. For example, once activation values for a layer have been stored of the bulk memory, forward propagation can continue for a number of different layers in the neural network. The stored activation values can later be retrieved after forward propagation has proceeded and decompressed to be used in back propagation operations.

As additional training cycles are performed, the stored, compressed activation values can be decompressed and used during back propagation for additional training cycles. As the neural network is trained, activation values can be compressed and decompressed any desired number of times.

XIV. Example Computing Environment

FIG. **21** illustrates a generalized example of a suitable computing environment **2100** in which described embodiments, techniques, and technologies, including performing activation compression based on a performance metric, can be implemented.

The computing environment **2100** is not intended to suggest any limitation as to scope of use or functionality of the technology, as the technology may be implemented in diverse general-purpose or special-purpose computing environments. For example, the disclosed technology may be implemented with other computer system configurations, including hand held devices, multi-processor systems, programmable consumer electronics, network PCs, minicomputers, mainframe computers, and the like. The disclosed technology may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

With reference to FIG. **21**, the computing environment **2100** includes at least one processing unit **2110** and memory **2120**. In FIG. **21**, this most basic configuration **2130** is included within a dashed line. The processing unit **2110** executes computer-executable instructions and may be a real or a virtual processor. In a multi-processing system, multiple processing units execute computer-executable instructions to increase processing power and as such, multiple processors can be running simultaneously. The memory **2120** may be volatile memory (e.g., registers, cache, RAM), non-volatile memory (e.g., ROM, EEPROM, flash memory, etc.), or some combination of the two. The memory **2120** stores software **2180**, images, and video that can, for example, implement the technologies described herein. A computing environment may have additional features. For example, the computing environment **2100** includes storage **2140**, one or more input devices **2150**, one or more output devices **2160**, and one or more communication connections **2170**. An interconnection mechanism (not shown) such as a bus, a controller, or a network, interconnects the components of the computing environment **2100**. Typically, operating system software (not shown) provides an operating environment for other software executing in the computing environment **2100**, and coordinates activities of the components of the computing environment **2100**.

The storage **2140** may be removable or non-removable, and includes magnetic disks, magnetic tapes or cassettes, CD-ROMs, CD-RWs, DVDs, or any other medium which can be used to store information and that can be accessed within the computing environment **2100**. The storage **2140** stores instructions for the software **2180**, which can be used to implement technologies described herein.

The input device(s) **2150** may be a touch input device, such as a keyboard, keypad, mouse, touch screen display, pen, or trackball, a voice input device, a scanning device, or another device, that provides input to the computing environment **2100**. For audio, the input device(s) **2150** may be a sound card or similar device that accepts audio input in analog or digital form, or a CD-ROM reader that provides audio samples to the computing environment **2100**. The output device(s) **2160** may be a display, printer, speaker, CD-writer, or another device that provides output from the computing environment **2100**.

The communication connection(s) **2170** enable communication over a communication medium (e.g., a connecting network) to another computing entity. The communication medium conveys information such as computer-executable instructions, compressed graphics information, video, or other data in a modulated data signal. The communication connection(s) **2170** are not limited to wired connections (e.g., megabit or gigabit Ethernet, Infiniband, Fibre Channel over electrical or fiber optic connections) but also include wireless technologies (e.g., RF connections via Bluetooth, WiFi (IEEE 802.11a/b/n), WiMax, cellular, satellite, laser, infrared) and other suitable communication connections for providing a network connection for the disclosed quantization-enabled computing systems. In a virtual host environment, the communication(s) connections can be a virtualized network connection provided by the virtual host.

Some embodiments of the disclosed methods can be performed using computer-executable instructions implementing all or a portion of the disclosed technology in a computing cloud **2190**. For example, the disclosed methods can be executed on processing units **2110** located in the computing environment **2130**, or the disclosed methods can be executed on servers located in the computing cloud **2190**.

Computer-readable media are any available media that can be accessed within a computing environment **2100**. By way of example, and not limitation, with the computing environment **2100**, computer-readable media include memory **2120** and/or storage **2140**. As should be readily understood, the term computer-readable storage media includes the media for data storage such as memory **2120** and storage **2140**, and not transmission media such as modulated data signals.

XV. Additional Examples of the Disclosed Technology

Additional examples of the disclosed subject matter are discussed herein in accordance with the examples discussed above.

A system of one or more computers can be configured to perform particular operations or actions by virtue of having software, firmware, hardware, or a combination of them installed on the system that in operation causes or cause the system to perform the actions. One or more computer programs can be configured to perform particular operations or actions by virtue of including instructions that, when executed by data processing apparatus, cause the apparatus to perform the actions. One general aspect includes a computing system including: one or more processors. The computing system also includes memory (bulk memory or other forms of memory) including computer-readable storage devices and/or memory. The computing system also includes a floating-point compressor formed from at least one of the processors, the floating-point compressor being in communication with the bulk memory; and the computing system being configured to produce a neural network including activation values expressed in a first floating-point format. The computing system can also be configured to select a second floating-point format for the neural network based on a performance metric for the neural network. The computing system can also be configured to convert at least one of the activation values to the second floating-point format, thereby producing compressed activation values. The computing system can also be configured to, with at least one of the processors, store the compressed activation values in the bulk memory. Other embodiments of this aspect include corresponding computer systems, apparatus, and computer programs recorded on one or more computer storage devices, each configured to perform the actions of the methods. The corresponding computer systems, apparatus, and computer programs can be used to perform any of the practical applications disclosed herein.

Implementations may include one or more of the following features. The computing system where the second floating-point format has a lower-precision mantissa than the first floating-point format. The computing system being further configured to: convert at least one of the compressed activation values to the first floating-point format to produce uncompressed activation values. The computing system may also be configured to perform backward propagation for at least one layer of the neural network with the uncompressed activation values, producing a further trained neural network. The computing system can be further configured to: perform forward propagation for the further trained neural network, producing updated activation values. The computing system can also be configured to based on the updated activation values, determine an updated performance metric. The computing system may also be configured to, based on the updated performance metric, select a third floating-point format different than the second floating-point format. The computing system may also be configured to converting at least one of the updated activation values to the third floating-point format to produce second compressed activation values. The computing system may also be configured to store the second compressed activation values in the bulk memory. In some examples of the computing system, the third floating-point format has a higher-precision mantissa than the second floating-point format. In some examples of the computing system, the second floating-point format has at least one mantissa expressed in an outlier mantissa format or a non-uniform mantissa format. Some examples, the computing system is further configured to: determine differences between an output of a layer of the neural network from an expected output. The computing system can be furthered configured to, based on the determined differences, select a third floating-point format by increasing a number of mantissa bits used to store the compressed activation values. In some examples of the computing system, the performance metric is based on at least one of the following for a layer of the neural network: number of true positives, number of true negatives, number of false positives, or number of false negatives. In some examples of the computing system, the first and second floating-point formats are block floating-point formats, the second floating-point format has a different sharing format of a common exponent than the first floating-point format, and the sharing format is different based on per-row, per-column, or per-tile sharing of a common exponent for the compressed activation values. In some examples of the computing system, the compressor is further configured to further compress the compressed activation values prior to the storing by performing at least one or more of the following: entropy compression, zero compression, run length encoding, compressed sparse row compression, or compressed sparse column compression. In some examples, the computing system is further configured to: perform backward propagation for a layer of the neural network by converting the stored, compressed activation values to activation values in the first floating-point format to produce uncompressed activation values. The computing system can also include perform a gradient operation with the uncompressed activation values. In some examples of the computing system, the processors include at least one of the following: a tensor processing unit, a neural network accelerator, a graphics processing unit, or a processor implemented in a reconfigurable logic array. The computing system can also include the bulk memory situated on a different integrated circuit than the processors. In some examples, the bulk memory includes dynamic random access memory (DRAM) or embedded DRAM and the system further includes a hardware accelerator including a memory temporarily storing the first activation values for at least a portion of only one layer of the neural network, the hardware accelerator memory including static RAM (SRAM) or a register file. Implementations of the described techniques may include hardware, a method or process, or computer software on a computer-accessible medium. The corresponding hardware, method or process, or computer software can be used perform any of the practical applications disclosed herein.

One general aspect includes a method of operating a computing system implementing a neural network, the method including, with the computing system, selecting an activation compression format based on a training performance metric for the neural network. The method also includes storing compressed activation values for the neural network expressed in the activation compression format in a computer-readable memory or storage device. Other embodiments of this aspect include corresponding computer systems, apparatus, and computer programs recorded on one or more computer storage devices, each configured to perform the actions of the methods. The method can be used perform any one or more of the disclosed practical applications.

Implementations may include one or more of the following features. The method further including: producing the training performance metric by calculating accuracy or change in accuracy of at least one layer of the neural network. The method further including: uncompressing the stored compressed activation values and training the neural network. The method may also include evaluating the training performance metric for the trained neural network. The method may also include based on the training performance metric, adjusting the activation compression format to an increased precision. The method may also include storing activation values for at least one layer of the trained neural network in the adjusted activation compression form. Examples of the method, the adjusted activation compression format has a greater number of mantissa bits than the selected activation compression format. In some examples of the method, the adjusted activation compression format has a different exponent format than the selected activation compression format. In some examples of the method, the selected activation compression format includes an outlier mantissa and the adjusted activation compression format does not include an outlier mantissa. In some examples of the method, the selected activation compression format

includes a non-uniform mantissa. In some examples, the selected activation compression format is a block floating-point format and the adjusted activation compression format is a normal-precision floating point format. In some examples, the training performance metric is based at least in part on one or more of the following for the neural network: a true positive rate, a true negative rate, a positive predictive rate, a negative predictive value, a false negative rate, a false positive rate, a false discovery rate, a false omission rate, or an accuracy rate. In some examples of the method, the training performance metric is based at least in part on one or more of the following for at least one layer of the neural network: mean square error, perplexity, gradient signal to noise ratio, or entropy. In some examples, the activation compression format has one of the following mantissa formats: lite lossy format, normal lossy format, or aggressive lossy format. One or more computer-readable storage devices or media storing computer-executable instructions, which when executed by a computer, cause the computer to perform the method. Implementations of the described techniques may include hardware, a method or process, or computer software on a computer-accessible medium. The corresponding hardware, method or process, or computer software can be used perform any one or more disclosed practical applications.

One general aspect includes a quantization-enabled system including: at least one processor. In some examples, the quantization-enabled system also includes a special-purpose processor configured to implement neural network operations in quantized number formats. The quantization-enabled system also includes a memory (e.g., bulk memory or other suitable memory) configured to received compressed activation values from the special-purpose processor. Other embodiments of this aspect include corresponding computer systems, apparatus, and computer programs recorded on one or more computer storage devices, each configured to perform the actions of the methods.

One general aspect includes a quantization-enabled system including: means for compressing neural network activation values produced during neural network training according to an accuracy parameter. The quantization-enabled system also includes means for storing the compressed neural network activation values. The quantization-enabled system also includes means for evaluating the neural network during the neural network training and, based on the evaluating, adjusting the accuracy parameter. Other embodiments of this aspect include corresponding computer systems, apparatus, and computer programs recorded on one or more computer storage devices, each configured to perform the actions of the methods.

Implementations may include one or more of the following features. The quantization-enabled system where the means for compressing includes: means for converting activation values from a first floating-point format to a second-floating-point format according to the accuracy parameter. The quantization-enabled system where the means for compressing includes: means for expressing the compressed neural network activation values in a lossy mantissa format. The quantization-enabled system where the means for compressing includes: means for expressing the compressed neural network activation values in an outlier-quantized format. Implementations of the described techniques may include hardware, a method or process, or computer software on a computer-accessible medium.

In view of the many possible embodiments to which the principles of the disclosed subject matter may be applied, it should be recognized that the illustrated embodiments are only preferred examples and should not be taken as limiting the scope of the claims to those preferred examples. Rather, the scope of the claimed subject matter is defined by the following claims. We therefore claim as our invention all that comes within the scope of these claims.

What is claimed is:

1. A method, implemented in a computing system comprising at least one hardware processor and at least one memory or storage device coupled to the at least one hardware processor, the method comprising, during a plurality of sets of epochs of training for a neural network:

setting a first activation compression format for the neural network, the neural network comprising at least a thousand interconnected nodes, each node comprising an activation function and a set of weights;

performing a first set of one or more epochs of training of the plurality of sets of epochs of training for the neural network using the first activation compression format, wherein an epoch of training comprises submitting inputs of a set of inputs to the neural network, determining an error term, and backpropagating the error term to the nodes of the neural network to update respective sets of weights of the neural network;

without user intervention, determining a first value for a training performance metric based on analysis of outputs derived from respective inputs in a set of training data for an epoch, wherein the training performance metric assesses the neural network's response characteristics relative to the respective inputs;

based at least in part on the first value for the training performance metric, by the computing system, without user intervention, determining that accuracy of the neural network, as indicated by the first training performance metric, has not increased or is improving by less than a determined amount;

based at least in part on determining that the accuracy of the neural network has not increased or is improving by less than a determined amount, by the computing system, without user intervention, selecting a second activation compression format having a higher precision than the first activation compression format;

performing a second set of one or more epochs of training of the plurality of sets of epochs of training for the neural network; and

during the performing a second set of one or more epochs of training, storing compressed activation values for the neural network expressed in the second activation compression format to provide stored compressed activation values.

2. The method of claim **1**, wherein the determining a first value of the training performance metric for the neural network comprises

calculating accuracy or change in accuracy of at least one layer of the neural network.

3. The method of claim **1**, further comprising:

determining a second value for the training performance metric after the second set of one or more epochs of training;

based on the second value for the training performance metric, adjusting the activation compression format to provide an adjusted activation compression format providing increased precision; and

storing activation values for at least one layer of the neural network in the adjusted activation compression form.

4. The method of claim **1**, wherein:

the second activation compression format has a greater number of mantissa bits than the first activation compression format; or

the first activation compression format is a block floating-point format and the second activation compression format is a normal-precision floating point format.

5. The method of claim **1**, wherein the first activation compression format comprises an outlier mantissa and the second activation compression format does not comprise an outlier mantissa.

6. The method of claim **1**, wherein the first activation compression format comprises a non-uniform mantissa.

7. The method of claim **1**, wherein the training performance metric is based at least in part on one or more of the following for the neural network: a true positive rate, a true negative rate, a positive predictive rate, a negative predictive value, a false negative rate, a false positive rate, a false discovery rate, a false omission rate, or an accuracy rate.

8. The method of claim **1**, wherein the training performance metric is based at least in part on one or more of the following for at least one layer of the neural network: mean square error, perplexity, gradient signal to noise ratio, or entropy.

9. A computing system comprising:

at least one hardware processor;

at least memory or storage device coupled to the at least one hardware processor; and

computer-readable instructions or hardware logic that, when executed, cause the computing system to perform operations during a plurality of sets of epochs of training for a neural network, the operations comprising:

setting a first activation compression format for the neural network, the neural network comprising at least a thousand interconnected nodes, each node comprising an activation function and a set of weights;

performing a first set of one or more epochs of training of the plurality of sets of epochs of training for the neural network using the first activation compression format, wherein an epoch of training comprises submitting inputs of a set of inputs to the neural network, determining an error term, and backpropagating the error term to the nodes of the neural network to update respective sets of weights of the neural network;

without user intervention, determining a first value for a training performance metric based on analysis of outputs derived from respective inputs in a set of training data for an epoch, wherein the training performance metric assesses the neural network's response characteristics relative to the respective inputs;

based at least in part on the first value for the training performance metric, by the computing system, without user intervention, determining that accuracy of the neural network, as indicated by the first training performance metric, has not increased or is improving by less than a determined amount;

based at least in part on determining that the accuracy of the neural network has not increased or is improving by less than a determined amount, by the computing system, without user intervention, selecting a second activation compression format having a higher precision than the first activation compression format;

performing a second set of one or more epochs of training of the plurality of sets of epochs of training for the neural network; and

during the performing a second set of one or more epochs of training, storing compressed activation values for the neural network expressed in the second activation compression format.

10. The apparatus of claim **9**, the operations further comprising:

producing values for the training performance metric by calculating accuracy or change in accuracy of at least one layer of the neural network.

11. The apparatus of claim **9**, the operations further comprising

uncompressing the stored compressed activation values and training the neural network;

evaluating the training performance metric for the trained neural network;

based on the training performance metric, adjusting the activation compression format to an increased precision; and

storing activation values for at least one layer of the trained neural network in the adjusted activation compression form.

12. The apparatus of claim **9**, wherein:

the second activation compression format has a greater number of mantissa bits than the first activation compression format; and

the first activation compression format is a block floating-point format and the second activation compression format is a normal-precision floating point format.

13. The apparatus of claim **9**, wherein the selected activation compression format comprises a non-uniform outlier mantissa and the adjusted activation compression format does not comprise an outlier mantissa.

14. The apparatus of claim **9**, wherein the training performance metric is based at least in part on one or more of the following for at least one layer of the neural network: a true positive rate, a true negative rate, a positive predictive rate, a negative predictive value, a false negative rate, a false positive rate, a false discovery rate, a false omission rate, or an accuracy rate, mean square error, perplexity, gradient signal to noise ratio, or entropy.

15. The apparatus of claim **9**, the operations further comprising:

producing a neural network comprising uncompressed activation values expressed in a first floating-point format, the first floating-point format having higher precision than the activation compression format;

selecting a second floating-point format for the neural network based on a value of the training performance metric; and

converting at least one of the activation values to the activation compression format, thereby producing the compressed activation values.

16. One or more computer-readable storage media or hardware logic comprising:

computer-executable instructions or logic that, when executed by a computing system comprising at least one memory or storage device and at least one hardware processed coupled to the at least one memory or storage device, cause the computing system to, during a plurality of sets of epochs of training for a neural network, set a first activation compression format for the neural network, the neural network comprising at least a thousand interconnected nodes, each node comprising an activation function and a set of weights;

computer-executable instructions or logic that, when executed by the computing system, cause the computing system to perform a first set of one or more epochs of training of the plurality of sets of epochs of training for the neural network using the first activation compression format, wherein an epoch of training comprises submitting inputs of a set of inputs to the neural network, determining an error term, and backpropagating the error term to the nodes of the neural network to update respective sets of weights of the neural network;

computer-executable instructions or logic that, when executed by the computing system, cause the computing system to, without user intervention, determine a first value for a training performance metric based on analysis of outputs derived from respective inputs in a set of training data for an epoch, wherein the training performance metric assesses the neural network's response characteristics relative to the respective inputs;

computer-executable instructions or logic that, when executed by the computing system, cause the computing system to, based at least in part on the first value for the training performance metric, without user intervention, determine that accuracy of the neural network, as indicated by the first training performance metric, has not increased or is improving by less than a determined amount;

computer-executable instructions or logic that, when executed by the computing system, cause the computing system to, without user intervention, based at least in part on determining that the accuracy of the neural network has not increased or is improving by less than a determined amount, select a second activation compression format having a higher precision than the first activation compression format;

computer-executable instructions or logic that, when executed by the computing system, cause the computing system to perform a second set of one or more epochs of training of the plurality of sets of epochs of training for the neural network; and

computer-executable instructions or logic that, when executed by the computing system, cause the computing system to, during the performing a second set of one or more epochs of training, store compressed activation values for the neural network expressed in the second activation compression format to provide stored compressed activation values.

17. The one or more computer-readable storage media or hardware logic of claim **16**, further comprising:

computer-executable instructions or logic that, when executed by the computing system, cause the computing system to determine a second value for the training performance metric after the second set of one or more epochs of training;

computer-executable instructions or logic that, when executed by the computing system, cause the computing system to, based on the second value for the training performance metric, adjust the activation compression format to provide an adjusted activation compression format providing increased precision; and

computer-executable instructions or logic that, when executed by the computing system, cause the computing system to store activation values for at least one layer of the neural network in the adjusted activation compression form.

18. The one or more computer-readable storage media or hardware logic of claim **16**, wherein:

the second activation compression format has a greater number of mantissa bits than the first activation compression format; or

the first activation compression format is a block floating-point format and the second activation compression format is a normal-precision floating point format.

19. The one or more computer-readable storage media or hardware logic of claim **16**, wherein the first activation compression format comprises an outlier mantissa and the second activation compression format does not comprise an outlier mantissa.

20. The one or more computer-readable storage media or hardware logic of claim **16**, wherein the first activation compression format comprises a non-uniform mantissa.

21. The one or more computer-readable storage media or hardware logic of claim **16**, wherein the training performance metric is based at least in part on:

(1) one or more of the following for the neural network: a true positive rate, a true negative rate, a positive predictive rate, a negative predictive value, a false negative rate, a false positive rate, a false discovery rate, a false omission rate, an accuracy rate; or

(2) one or more of the following for at least one layer of the neural network: mean square error, perplexity, gradient signal to noise ratio, or entropy.

* * * * *